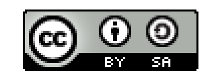




Diseño e Implementación de Bases de Datos desde una Perspectiva Práctica



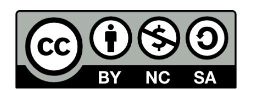
AUTORES

Héctor Cardona
Jhon Eder Masso
Maritza Fernanda Mera
Sandra Milena Roa
Edgar Fabián Ruano
María Dolores Torres
María Isabel Vidal

Diseño e Implementación de Bases de Datos desde una Perspectiva Práctica

1a ed. - Iniciativa Latinoamericana de Libros de Texto Abiertos (LATIn), 2014. 147 pag.

Primera Edición: Marzo 2014
Iniciativa Latinoamericana de Libros de Texto Abiertos (LATIn)
http://www.proyectolatin.org/









Este texto forma parte de la Iniciativa Latinoamericana de Libros de Texto abiertos (LATIn), proyecto financiado por la Unión Europea en el marco de su Programa ALFA III EuropeAid.
El Proyecto LATIn está conformado por: Escuela Superior Politécnica del Litoral, Ecuador (ESPOL); Universidad Autónoma de Aguascalientes, México (UAA), Universidad Católica de San Pablo, Perú (UCSP); Universidade Presbiteriana Mackenzie, Brasil(UPM); Universidad de la República, Uruguay (UdelaR); Universidad Nacional de Rosario, Argentina(UNR); Universidad Central de Venezuela, Venezuela (UCV), Universidad Austral de Chile, Chile (UACH), Universidad del Cauca, Colombia (UNICAUCA), Katholieke Universiteit Leuven, Bélgica (KUL), Universidad de Alcalá, España (UAH), Université Paul Sabatier, Francia (UPS).

Índice general

1 INTRODUCCIÓN A LOS MODELOS DE DATOS 9

1.1 Definición de modelos de datos 9

1.2 Componentes de un Modelo de Datos 11

1.2.1 Estática 11
1.2.2 Dinámica 11

1.3 Tipos de Modelos de Datos 11

1.3.1 Modelo Externo 13
1.3.2 Modelos Globales 13
1.3.3 Modelos Internos 15

1.4 METODOLOGIA DE DISEÑO DE BASES DE DATOS 15

1.4.1 Definición de metodología de diseño y desarrollo de bases de datos 15
1.4.2 Diseño Conceptual 16
1.4.3 Diseño Lógico 17
1.4.4 Diseño Físico 18
1.4.5 Ejemplo 18

2 MODELO CONCEPTUAL: ENTIDAD INTERRELACIÓN 21

2.1 Introducción 21

2.2 Componente Estático 21

2.2.1 Entidad 21
2.2.2 Atributo 23
2.2.3 Dominio 26
2.2.4 Relación 27

2.3 Semántica de las Interrelaciones 28

2.3.1 Nombre de la Relación 28
2.3.2 Grado de la relación 28
2.3.3 Correspondencia y cardinalidad de las interelaciones 29

2.4 Mecanismos de Abstracción 31

2.4.1 Clasificación 31
2.4.2 Generalización 32

2.5 Modelo Entidad Relación Extendido 33

2.5.1 Restricciones sobre interrelaciones 33
2.5.2 Generalización 35
2.5.3 Agregación 37

3 MODELO LÓGICO: RELACIONAL 41

3.1 INTRODUCCIÓN 41

3.2 TRANSFORMACIÓN DEL MODELO ENTIDAD-RELACIÓN (E-R) A MODELO RELACIONAL 41

3.2.1 Cardinalidad 1:1 42
3.2.2 Cardinalidad 1:N 43
3.2.3 Cardinalidad N:M 44

3.3 NORMALIZACIÓN 44

3.3.1 Relación 44
3.3.2 Razones para normalizar 46
3.3.3 Formas Normales 48

4 CAPITULO IV - MODELO FÍSICO 65

4.1 ÁLGEBRA RELACIONAL 65

4.1.1 Operaciones Unarias 65
4.1.2 Operaciones Binarias 69
4.1.3 Operaciones de Agregación 78

4.2 STRUCTURED QUERY LANGUAGE (SQL) - DATA DEFINITION LANGUAGE (DDL) 81

4.2.1 Tipos de Datos Estandar 81
4.2.2 Sentencias de Creación 82
4.2.3 Sentencia de Modificación 84

4.3 STRUCTURED QUERY LANGUAGE (SQL) - DATA MANIPULATION LANGUAGE (DML) 85

4.3.1 Consultas 85
4.3.2 Sub-Consultas 100
4.3.3 Inserción 103
4.3.4 Borrado 106
4.3.5 Actualización 107

4.4 USUARIOS, ROLES Y PRIVILEGIOS 109

4.4.1 Contextualización 109
4.4.2 Usuarios 110
4.4.3 Roles 111
4.4.4 Privilegios 111
4.4.5 Asignar y revocar privilegios 111
4.4.6 Ejemplo práctico 112

5 APÉNDICE A. Ejemplo de diseño e implementación de una base de datos relacinal en POSGRESQL 115

5.1 A.1. PLANTEAMIENTO DEL CASO 115

5.2 A.2. DISEÑO CONCEPTUAL 116

5.3 A.3. DISEÑO LÓGICO 116

5.4 A.4. ESQUEMA FÍSICO DE LA BASE DE DATOS 117

6 APÉNDICE B. Ejemplo de diseño e implementación de una base de datos relacional en ORACLE 123

6.1 B.1. PLANTEAMIENTO DEL CASO 123

6.2 B.2. DISEÑO CONCEPTUAL 125

6.3 B.3. DISEÑO LÓGICO 126

6.4 B.4. IMPLEMENTACION ORACLE 126

6.4.1 Código SQL correspondiente al modelo relacional de base de datos 126

6.4.2 Código SQL para manejar restricciones propias del problema modelado . 135

7 APENDICE C. Ejemplo de diseño e implementación de una base de datos relacional 139

7.1 C.1. PLANTEAMIENTO DEL CASO 139

7.2 C.2. DISEÑO CONCEPTUAL 140

7.3 C.3. DISEÑO LOGICO 140

7.4 C.4. IMPLEMENTACION 141

1 — INTRODUCCIÓN A LOS MODELOS DE DATOS

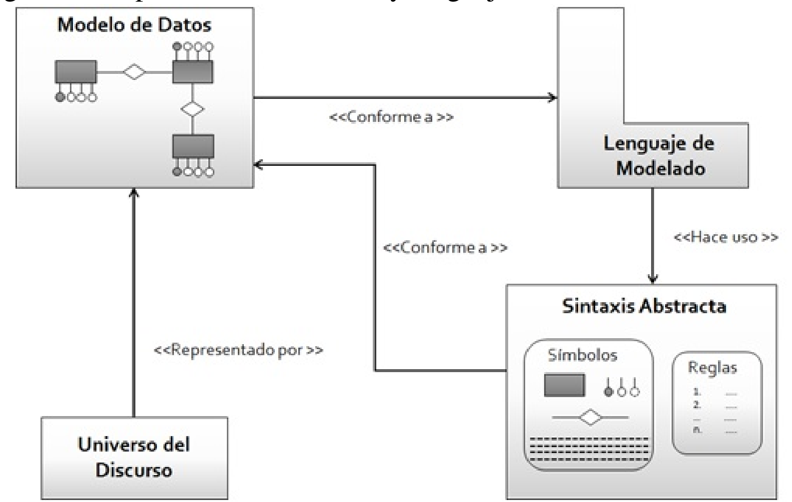
1.1 Definición de modelos de datos

Los modelos de datos son mecanismos que permiten la abstracción y representación de un dominio, mediante un conjunto de reglas y símbolos pertenecientes a un lenguaje de modelado que es conforme al modelo. Un modelo puede definirse como la abstracción que permite representar los diferentes elementos pertenecientes a un dominio del negocio, sus relaciones y asociaciones, conforme sucedería en el mundo real.

Los modelos pueden entenderse como las piezas claves que permiten describir y especificar un universo del discurso, es decir representar una realidad en concreto. Por tanto, estos modelos consideran un conjunto de elementos claves los cuales permitirán describir algo físico, abstracto o una realidad hipotética.

Un modelo es una representación simplificada de una realidad existente, expresada bajo la utilización de un lenguaje de modelado bien definido que hace uso de una sintaxis abstracta propia del modelo que permite la representación y especificación de un dominio en particular. La sintaxis abstracta estará conformada por los símbolos y el conjunto de reglas definidas para dichos elementos. Esta representación queda resumida en la figura 1.1.

Figura 1.1: Aplicación de modelos y lenguajes a un universo del discurso



La importancia de los modelos radica en que permiten representar un universo del discurso o dominio del negocio haciendo uso de la abstracción, a fin de identificar aspectos importantes y relevantes para el dominio, esta representación será posible con la ayuda de un lenguaje de modelado, a fin de entender y comprender un problema complejo que brinde una solución que dé respuesta a una necesidad.

El universo del discurso (UD) podría definirse como el conjunto de necesidades de un

dominio de negocio en particular, que nos permite el modelado e implementación de una base de datos. En otras palabras, es una descripción clara y precisa sobre el universo o mundo que se desea modelar.

Un universo del discurso es realizado por expertos o con la ayuda de estos, ya que estos son los que entienden del negocio; es decir los expertos en la materia. Podríamos decir que un UD es una descripción del dominio del negocio en términos de un experto que permite la compresión y la realización del modelado de una base de datos.

Ejemplo de un Universo del Discurso

Figura 1.2: Universo del discurso



Fuente: `http://www.cristiancaricaturas.com/`

"Se desea construir una base de datos para la gestión de una empresa dedicada a la venta de insumos para la contrucción y alquiler de andamios, la cual esta ubicada en la ciudad de Popayán. La empresa desea controlar toda la información de ventas, alquiles, empleados, clientes y de cada una de sus sedes que estan ubicadas por toda la ciudad. Con respecto a las sedes es preciso tener en cuenta que tienen un identificador principal, dirección, telefono y fax , además estas deberán estar a cargo de un empleado y un empleado podrá estar a cargo de estas pero en un determinado tiempo, es decir un empleado no puede tener a cargo más de una sede en un periodo de tiempo......... "

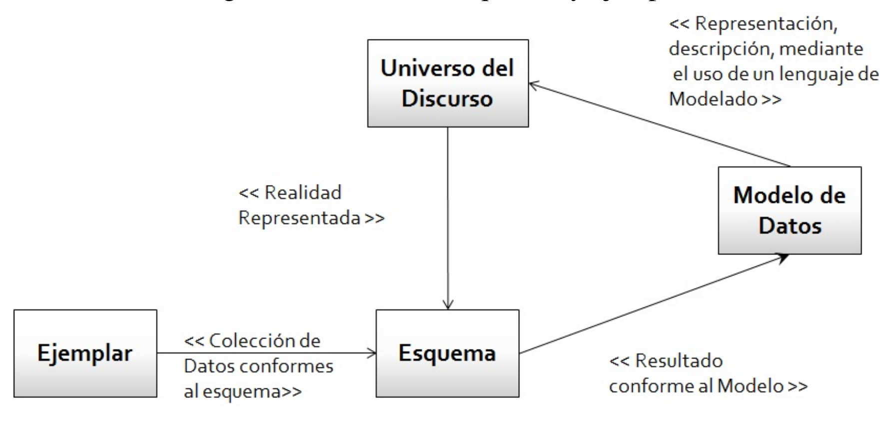
Es importante distinguir en el modelado de datos los conceptos de Esquema y Ejemplar.

Esquema: Es la realidad representada mediante la utilización de modelos en diferentes niveles de abstracción del proceso de construcción de una base de datos; es aquí en donde se encuentra la estructura de una base de datos que representa los conceptos e instancias de cada uno de esos elementos que han sido modelados.

Ejemplar: Entendido como una ocurrencia o instancia de la estructura de la base de datos, es decir, colección de datos dinámicos que están almacenados y representados con forme a un esquema en un determinado momento.

Podemos concluir con respecto a los modelos de datos que estos hace uso de un conjunto de reglas, restricciones y símbolos que han sido definidos para un lenguaje de modelado, con el fin de representar la semántica de un UD a fin de crear un esquema que permita coleccionar instancias de cada uno de los elementos pertenecientes a un dominio que fue modelado. Ver Figura 1.3.

Figura 1.3: Modelos, esquemas y ejemplares



1.2 Componentes de un Modelo de Datos

Los modelos de datos están expresados mediante dos propiedades, las cuales son estáticas y dinámicas. Las estáticas hacen referencia a la estructura del modelo. Las dinámicas al conjunto de instancias o valores que toman cada uno de los componentes de un modelo y a las operaciones sobre los mismos.

1.2.1 Estática

Esta propiedad es la que define el conjunto de símbolos, reglas y restricciones de un modelo. La estática de un modelo es lo que permite representar:

- **Objetos,** es decir entidades o conceptos del UD que deseamos representar.
- **Propiedades** o características propias de los objetos, también conocidos como atributos.
- **Relaciones** o asociaciones entre cada uno de los objetos que intervienen en el modelo.
- **Y restricciones** o elementos no permitidos, es decir las limitaciones o reglas de integridad definidas para los modelo de datos y las que son propias al UD, estas últimas son las definidas por los expertos del negocio y podrán ser aplicadas a nivel de los objetos y sus propiedades o en las relaciones entre los mismo objetos.

1.2.2 Dinámica

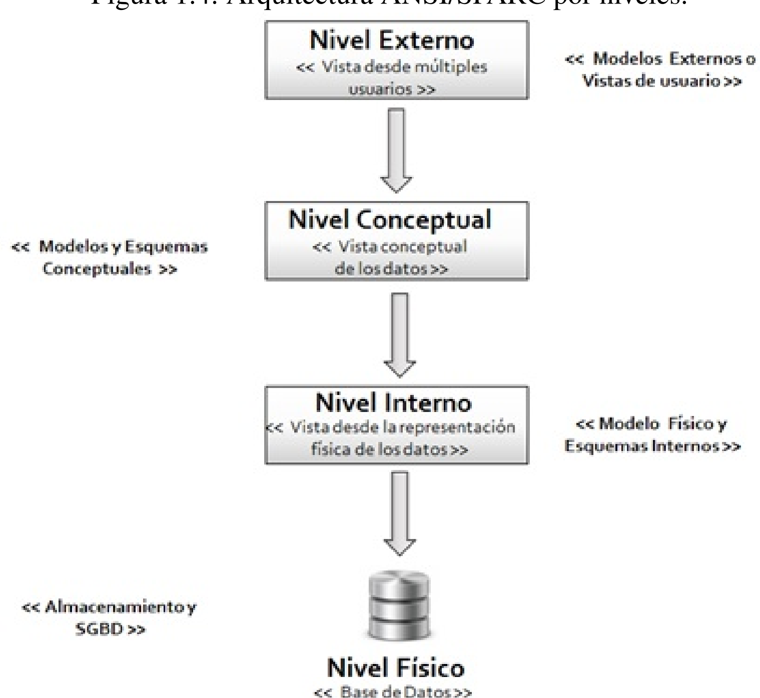
Esta propiedad comprende todas las operaciones que se pueden realizar sobre el conjunto de instancias de un esquema. El componente dinámico define un conjunto de operadores para la realización de operaciones entre objetos, sobre las propiedades de los objetos y entre otras operaciones.
Algunas de estas operaciones podrían ser de actualización o recuperación de información alojada en un esquema de base de datos y las cuales serán programadas por un usuario haciendo uso de lenguaje de consulta el cuál brinda las sentencias necesarias para acceder a la información.

1.3 Tipos de Modelos de Datos

A mediados de los 70's e inicios de los 80's el Instituto Nacional Estadounidense de Estándares (American National Standards Institute or ANSI) y el Comité de Requisitos y Planificación de Estándares (Standards Planning And Requirements Committee or SPARC), proponen una arquitectura o esquema de tres niveles, también conocido como interfaces de vistas para una base de datos. Estos niveles permiten pensar la base de datos desde la abstracción y representación de un UD, así como también la visión personalizada de la base de datos en términos de cada uno

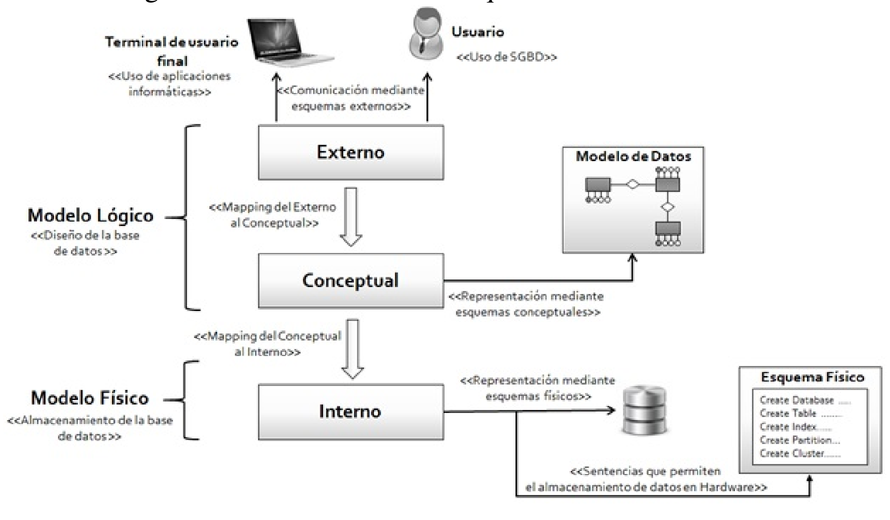
de los usuarios que intervienen en la creación, gestión y manipulación de este tipo de sistemas. Es decir, una interfaz describe información importante y relevante en términos de los usuarios involucrados en cada una de ellas. Ver Figura 1.4.

Figura 1.4: Arquitectura ANSI/SPARC por niveles.



Los niveles que fueron definidos en este estándar son: ***Externo, Conceptual*** e **Interno**; Respecto al ***nivel externo*** podemos decir que hace referencia a una vista de un usuario, la cual describe sola una parte de la base de datos la cual es relevante para el usuario. Esta vista limita al usuario a solo ver información que le ha sido autorizada, es decir, una vista que excluye todos datos que el usuario no le ha sido autorizado o no puede acceder. En este nivel es donde aparecen las interfaces de aplicaciones de usuario finales y los lenguajes de manipulación de datos.
El ***nivel conceptual***, consiste en la forma de representación cada uno de los elementos de un UD. Es aquí donde aparecen modelos que permiten la representación y descripción de los datos en términos de entidades, propiedades y relaciones entre las mismas, además de la integridad, seguridad y restricciones en los datos. Aquí podemos hablar de un esquema el cual está acorde a un modelo de datos, es decir, un conjunto formal de reglas que permiten la representación de los datos y las operaciones entre los mismos.
El ***nivel interno***, define la forma en como los datos serán representados y almacenados físicamente mediante un sistema informático o gestor de bases de datos, es decir, la definición de estructuras que permiten el almacenamiento de los datos en la base datos así como también en el hardware del equipo. En este nivel estaríamos hablando de vistas a nivel de los equipos computaciones y los sistemas informáticos que se encargan de la gestión de bases de datos, además del esquema físico que es la representación de una base de datos, en términos de un conjunto de sentencias que son entendidas por un sistema informático en particular. La Figura 1.5 muestra un resumen de cada uno de los elementos de esta arquitectura mencionados anteriormente.

Figura 1.5: Elementos de la Arquitectura ANSI/SPARC.



1.3.1 Modelo Externo

Hace referencia a la forma en como le son presentados los datos a los diferentes tipos de usuarios de la base de datos. Ya sea mediante la utilización de aplicaciones informáticas o mediante la utilización de un SGBD. Aquí hablamos de Vista Externa. Esta vista como se mencionó anteriormente limita a los usuarios a solo ver la información que le ha sido autorizada, además dependerá del ambiente o entorno en el cual se desenvuelve o trabaja el usuario. Ejemplo una unidad organizativa.

1.3.2 Modelos Globales

Son los modelos de datos que permiten la abstracción y representación de los datos. Estos corresponden a la vista conceptual que se propone en la arquitectura ANSI/SPARC. Estos están divididos en ***Conceptual*** y ***Lógico***.

Modelo Conceptual

Este modelo representa una vista abstracta y global de toda la base de datos, en términos de objetos, propiedades y relaciones. Es decir, describe un esquema básico del cómo se podrán tratar los datos en un futuro por el modelo interno. El modelo conceptual está más interesado en la naturaleza básica de la representación del conjunto de necesidades expresadas en el UD. Los modelos conceptuales son orientados más hacia el análisis de la base de datos que al diseño o implementación, por tanto estos modelos son independientes de herramientas informáticas, lo cual quiere decir que no dependen de ningún Sistema Gestor de Bases de Datos (SGBD), a esto es lo que se le conoce como independencia de los datos con respecto a los modelos externos e internos.Algunos de los principales modelos conceptuales para la representación de datos son:

- El ***Modelo Infológico*** (Infological) tiene como finalidad la representación de la información conforme a es percibida por las personas, en términos de colección de objetos + propiedades o relaciones + el tiempo, siendo cada uno de estos los elementos básicos del modelo. Finalmente este modelo hace más énfasis en los aspectos conceptuales y estructurales de los datos.
- El ***Modelo de Datos Semántico*** (Semantic Data Models - SDM) provee un conjunto de mecanismos que permite realizar un modelado de alto nivel orientado a capturar la semántica del entorno de la aplicación en términos de los diferentes tipos de entidades existente en ese entorno, la agrupación e interconexión estructural de las mismas.

- El ***Modelo [JMD3] [JMD4] Relacional / Tasmania*** (Relational Model/Tasmania - RM/T) es una versión extendida del modelo relacional. Contempla algunos de los principios del modelo Entidad/Relación. Este modelo permite representar las entidades y sus relaciones, junto a sus propiedades, además posee un conjunto de operadores especiales los cuales permitan la manipulación de los diferentes objetos del RM/T.
- El ***Modelo Entidad Relación MER***, es uno de los modelos conceptuales más utilizados. Permite crear una representación básica de una base de datos en términos de entidades, atributos o propiedades, relaciones entre entidades y restricciones o reglas que permiten la integridad de los datos. El esquema conceptual generado por este modelo es independiente de las herramientas que se utilizan para la implementación de bases de datos.
- ***Otros***.Finalmente, un modelo conceptual es una arquitectura de la base de datos en términos de un conjunto de símbolos que representan conceptos u objetos, propiedades y relaciones, las cuales son fundamentales para el negocio o ámbito de la base de datos.

Modelo Lógico

También conocidos modelos convesionales. Estos modelos describen la arquitectura de la base de datos con el fin de que pueda ser implementada por un SGBD. Es decir, estos modelos están asociados a las herramientas que permiten la implementación de bases de datos. Un modelo lógico permite realizar una descripción más detallada de las entidades, relaciones y sus propiedades. Algunos de los principales modelos lógicos son: Jerárquico, Codasyl y Relacional.

- El ***Modelo Jerárquico***, es un modelo en red que permite la representación de los datos mediante una estructura en árbol, donde los nodos del árbol representan todas las entidades o conceptos principales del UD y los arcos son las relaciones entre estas entidades. Este modelo presenta una serie de incapacidades a la hora de representar algunos hechos de la vida real como por ejemplo relaciones muchos a muchos, reflexivas, etc, lo cual genera que existan redundancias a la hora de modelar los datos.
- El ***Modelo Codasyl*** que recibe el nombre del Committee for Data Systems Languages, es un modelo de red el cual sigue una estructura en árbol que le permite a un al diseñador de bases de datos crear una red de esquemas en varios niveles. Este modelo permitió la creación SGBD, comúnmente conocidos como ***CODASYL DBMS***. Además describe una serie de lenguajes que permiten la definición de los esquemas y sub-esquemas de la base de datos, al igual que estructuras para la manipulación de los mismos.
- El ***Modelo Relacional*** es una propuesta basada en las teorías de conjuntos y el concepto matemático de relaciones. Este modelo pretende darle más importancia a la representación de los datos que al almacenamiento de los mismos. Es uno de los más extendidos a la hora de realizar el diseño y modelado de bases de datos. El modelo relacional y su esquema vendrá definido en términos de relaciones o tablas que serán las encargadas de representar y almacenar todos los objetos del UD junto a sus ocurrencias. Las relaciones deberán contener tener filas y columnas, las filas serán las ocurrencias de estos objetos en la base de datos y las columnas son las propiedades o atributos de la relación, además deberá incorporar una columna como identificador principal o llave primara que es lo que permitirá gestionar la integridad de las ocurrencias de relación y las llaves foráneas o externas que son las que especifican los diferentes vínculos o referencias entre las diferentes relaciones modeladas. Este modelo dio origen a los actuales SGBD relacionales.
- ***Otros***.Finalmente estos modelos permiten crear esquemas de base datos que son más cercanos para la implantación de la base de datos, es decir, son modelos que están más orientados hacia el diseño de una base de datos.

1.3.3 Modelos Internos

Estos modelos están más orientados a los SGBD. Estos permitirán la creación del modelo físico de la base de datos. Estos permiten la creación de los esquemas internos y físicos de la base de datos, además cumplirán con las características y especificaciones propias del SGBD que se escogió para la implementación.

Modelo Físico

Es un modelo que describe la abstracción de la base de datos al más bajo nivel, es decir, describe la forma en cómo serán almacenados los datos, esto dependerá finalmente del SGBD seleccionado para la implementación el cual es escogido sean las necesidades de los usuarios y del negocio. Un modelo físico incluirá todas las sentencias conforme a un lenguaje de definición de datos apropiado para la creación de todos los objetos de la base datos, las relaciones entre cada una de las tablas, así como también los índices, sinónimos, restricciones, clusters, etc. Cuando nos referimos al SGBD seleccionado por un usuario, se está haciendo hincapié a las diferentes bases de datos comerciales existentes en el mercado, como por ejemplo: Informix, Oracle, Postgres, SQL Server, Sybase, DB2, MySQL, etc.Finalmente, esto modelos están más orientados hacia la implementación de las bases de datos tomando como punto de partida cada una de las restricciones de los SGBD, afín de que puedan ser utilizadas por aplicaciones y de los diferentes usuarios de la base de datos.

1.4 METODOLOGIA DE DISEÑO DE BASES DE DATOS

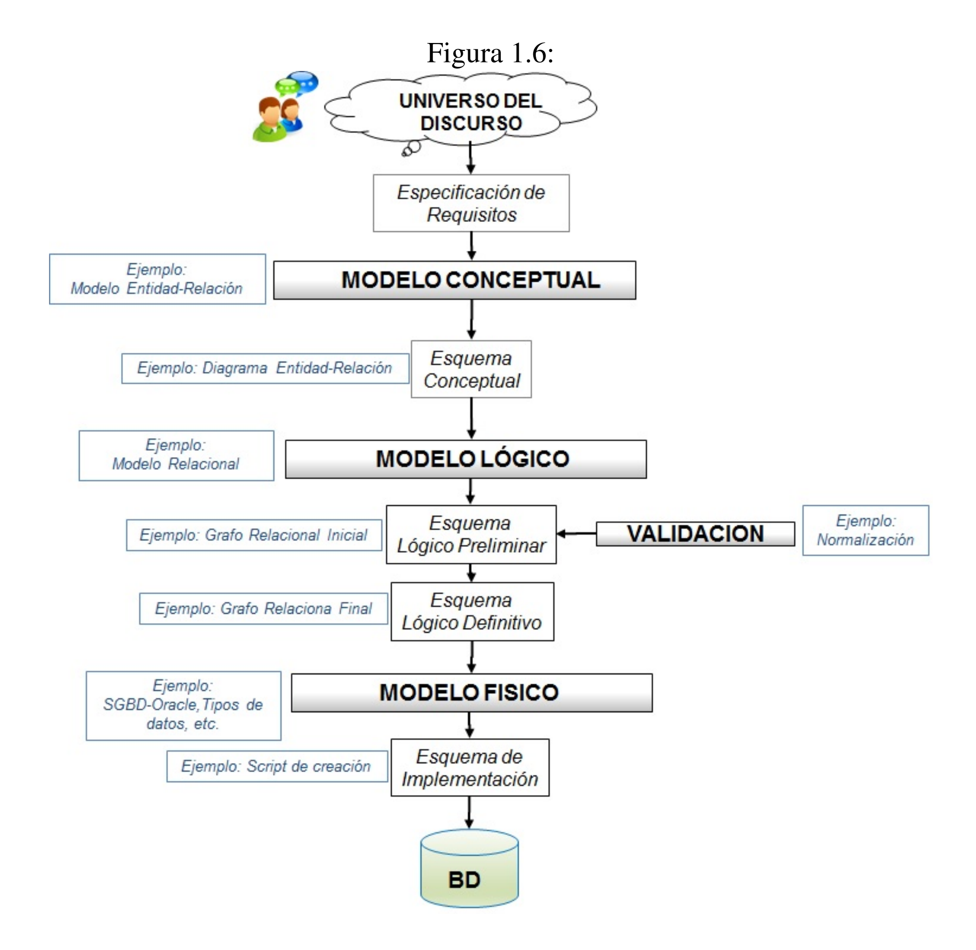
En este subcapítulo se definirán las fases de la metodología propuesta para el diseño y desarrollo de una base de datos desde el Universo del Discurso hasta su implementación en archivos físicos de datos.

1.4.1 Definición de metodología de diseño y desarrollo de bases de datos

Una metodología de diseño y desarrollo de bases de datos puede ser definida como un conjunto de procedimientos y técnicas agrupadas en etapas que guían al diseñador en el proceso de construcción de una base de datos facilitando a partir de los requisitos, necesidades o problemas de un usuario en términos de datos la concepción de una solución física soportada en una base de datos que le permita obtener la información requerida.

En la figura se presenta los principales elementos que deben considerarse en una metodología de diseño y desarrollo de un sistema de base de datos simple o complejo.

Figura 1.6:



1.4.2 Diseño Conceptual

En esta etapa llamada Diseño Conceptual, la entrada es la Especificación de los Requisitos del usuario, es decir, la información obtenida del Universo del Discurso o Semántica, donde el principal objetivo es la captura y Modelamiento de los “conceptos” que guiarán el diseño en términos del usuario entendidos por el diseñador mediante la selección y uso de un Modelo de Datos Conceptual como proceso de esta etapa, la salida de esta etapa es un Esquema Conceptual.

Este diseño es independiente de consideraciones físicas como el Sistema Gestor de Base de Datos entre otras.

Figura 1.7: Etapa de Diseño Conceptual.

Especificación de Requisitos → MODELO CONCEPTUAL → Esquema Conceptual

Actividades:

1. Identificación de los conceptos contenidos en la especificación de requisitos o universo del discurso asociados a:
   a) Características, propiedades o atributos. (generalmente adjetivos)
   b) Conjuntos de elementos que contienen características similares (generalmente sustantivos)
   c) Relaciones entre los elementos identificados. (generalmente verbos)

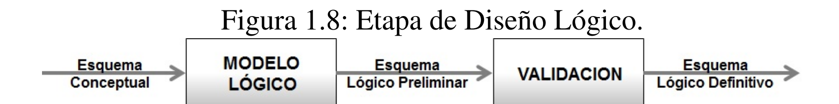
2. Revisar si existen valores que pueden tomar los atributos y en caso afirmativo, determinar la lista de los posibles valores, los cuales serán llamados en adelante dominios.
3. Determinar los atributos que identifican un conjunto de elementos que tienen características similares.
4. Seleccionar un Modelo de Datos Conceptual que permitan modelar todos los objetos encontrados.
5. Crear a partir del modelo de datos seleccionado el respectivo Esquema Conceptual de la base de datos.
6. Revisar el esquema obtenido garantizando que no exista redundancia de los datos, que refleje la semántica descrita por el usuario, que soporte transacciones de los usuarios, entre otros aspectos.
7. Presentar el Esquema Conceptual a los usuarios y realizar las correcciones necesarias para obtener el Esquema Definitivo acorde a los requisitos.
8. Documentar esta etapa considerando todos los elementos y las relaciones de tipo no estructural encontradas, es decir, aquellas que deben ser tenidas en cuenta pero que no se pueden representar por el Modelo de Datos seleccionado, incluir el diccionario de datos.

1.4.3 Diseño Lógico

En esta etapa de la metodología propuesta, la entrada es el Esquema Conceptual de la etapa anterior, a partir del cual y mediante la selección y aplicación de un Modelo de Datos Lógico se obtendrá un Esquema Lógico, en donde el objetivo de esta etapa es la transformación de los conceptos capturados del usuario que contemple crecimientos futuros y datos estructuralmente correctos que permitan realizar las transacciones requeridas por los usuarios.

Este diseño sigue siendo independiente de consideraciones físicas aunque los Sistemas Gestores de Bases de Datos tienen implementado o atienden a un Modelo de Datos específico.

Figura 1.8: Etapa de Diseño Lógico.



Actividades:

1. Seleccionar un Modelo de Datos Lógico e identificar sus estructuras, reglas y restricciones.
2. Transformar los elementos contemplados en el Esquema Conceptual según el Modelo Lógico seleccionado.
3. Identificar y determinar las restricciones asociadas al universo del discurso, entre ellas las de integridad, de identidad, etc.
4. Determinar los elementos no estructurales que deberán ser contemplados en la siguiente etapa del diseño físico que no pueden ser representadas por el Modelo Lógico Seleccionado.
5. Integrar en un esquema lógico global, en caso que existan varios esquemas lógicos, es decir, las diferentes vistas de los usuarios.
6. Validar el esquema lógico obtenido verificando que se conserve la semántica inicial contenida en la Especificación de Requisitos, que cumpla con las restricciones de integridad, de identidad y otras consideraciones como soportar las transacciones de los usuarios.
7. Documentar el esquema lógico obtenido estructuralmente y sus respectivas consideraciones de tipo no estructural.

1.4.4 Diseño Físico

El diseño físico es la etapa que contempla la traducción del Esquema Lógico obtenido en la etapa anterior en un Esquema Físico que pueda implementarse en un Sistema Gestor de Base de Datos Específico, es decir, existe una alta dependencia física con el sistema seleccionado.

La entrada de esta etapa son el Esquema Conceptual, su respectiva transformación y validación contemplada en el Esquema Lógico y la documentación asociada al seguimiento de esta metodología, y estos elementos permitirán obtener como salida una descripción o esquema de implementación de la base de datos en almacenamiento secundario considerando organización de archivos, índices, restricciones, etc.

Figura 1.9: Etapa Diseño Físico.

Esquema Lógico Definitivo → MODELO FISICO → Esquema de Implementación

Actividades:

1. Seleccionar el Sistema Gestor de Base de Datos en el cuál se implementará la base de datos e identificar sus funcionalidades.
2. Representar e implementar los elementos del Esquema Lógico en las estructuras permitidas por el Modelo Físico que sigue el Sistema Gestor, teniendo en cuenta todas las consideraciones estructurales y no estructurales.
3. Determinar la organización de los archivos y los índices requeridos según el análisis de los datos que serán almacenados y las transacciones previstas, en esta actividad debe considerarse la estimación del espacio en disco.
4. Implementar mecanismos de seguridad y control de acceso a los datos.
5. Garantizar el rendimiento de la base de datos y tomar acciones como incluir redundancia controlada en caso de ser necesario buscando mejorar el desempeño del sistema por ejemplo el tiempo de ejecución y/o recuperación en una consulta a la base de datos.
6. Documentar las consideraciones tenidas en cuenta para la implementación de la base de datos física.

1.4.5 Ejemplo

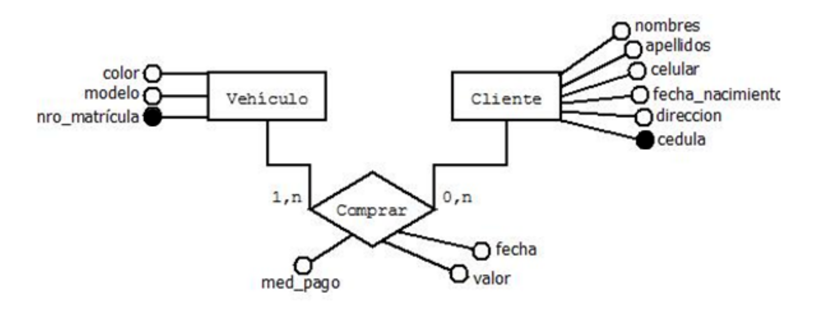
UNIVERSO DEL DISCURSO:
En un concesionario de vehículos se desea mediante una base de datos guardar la información relacionada con los autos y clientes, sabiendo que de los vehículo se tiene el número de matrícula, color y modelo; se consideran clientes a las personas que han realizado una compra de vehículo en el concesionario, guardándose los datos del cliente (cédula de ciudadanía, nombres, apellidos, dirección, celular, fecha de nacimiento) y con respecto a la compra se desea saber la fecha en que se realizó asimismo el valor y medio de pago utilizado (cheque, efectivo, tarjeta de crédito). Algunos de los clientes han realizado más de una compra en el concesionario.

1. Se realizará el diseño conceptual para lo cual se identificaran:
   a) Características: color, modelo, número de matrícula, cédula de ciudadanía, nombres, apellidos, fecha de nacimiento, dirección, celular, fecha compra, valor, medio de pago.

b) Conjuntos: Vehículos, Clientes

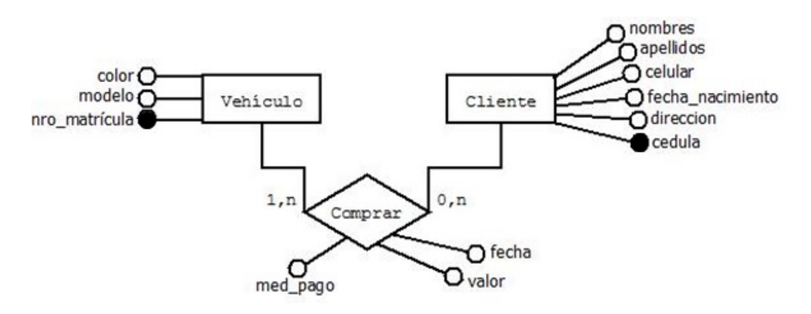
c) Relaciones: Comprar

d) Modelo de Datos Conceptual seleccionado: Entidad/Interrelación



e) Esquema Conceptual Inicial

f) Esquema Conceptual Definitivo y estructuras no contempladas a tener en cuenta



med_pago = {"efectivo", "cheque", "tarjeta"}

g)

2. Se realizará el diseño lógico teniendo en cuenta:
   a) Modelo de Datos Lógico seleccionado: Relacional
   b) Transformación del esquema conceptual a esquema lógico:

Vehículo (<u>nro_matricula</u>, color, modelo)
Cliente (<u>cedula</u>, dirección, fecha_nacimiento, celular, apellidos, nombres)
Comprar (med_pago, fecha, valor)

   c) Restricciones
   d) Elementos no estructurales considerados

3. La validación del esquema relacional se realizará con las Formas Normales o Teoría de la Normalización y el esquema relacional definitivo es:

Vehículo (<u>nro_matricula</u>, color, modelo)
Cliente (<u>cedula</u>, dirección, fecha_nacimiento, celular, apellidos, nombres)
Comprar (<u>cedula, nro_matricula</u>, med_pago, fecha, valor)

4. El diseño físico del ejemplo contempla:
   a) El SGBD seleccionado: Oracle

b) La implementación se realizará en PL/SQL siendo el siguiente el script correspondiente

```
/*==============================================================*/
/* Table: CLIENTE                                               */
/*==============================================================*/
create table CLIENTE (
  CEDULA            DEC(10)              not null,
  DIRECCION         VARCHAR(25),
  FECHA_NACIMIENTO  DATE,
  CELULAR           CHAR(10)             not null,
  APELLIDOS         VARCHAR(20)          not null,
  NOMBRES           VARCHAR(20)          not null,
  constraint PK_CLIENTE primary key (CEDULA)
);

/*==============================================================*/
/* Table: COMPRAR                                               */
/*==============================================================*/
create table COMPRAR (
  CEDULA            DEC(10)              not null,
  NRO_MATRICULA     CHAR(7)              not null,
  MED_PAGO          VARCHAR(12)
    constraint CKC_MED_PAGO_COMPRAR check (MED_PAGO is null or (MED_PAGO in ('Cheque','Tarjeta','Efectivo','Otro'))),
  FECHA             DATE                 not null,
  VALOR             DEC(12),
  constraint PK_COMPRAR primary key (CEDULA, NRO_MATRICULA)
);

/*==============================================================*/
/* Table: VEHICULO                                              */
/*==============================================================*/
create table VEHICULO (
  NRO_MATRICULA     CHAR(7)              not null,
  COLOR             VARCHAR(20),
  MODELO            INT,
  constraint PK_VEHICULO primary key (NRO_MATRICULA)
);

alter table COMPRAR
  add constraint FK_COMPRAR_REFERENCE_CLIENTE foreign key (CEDULA)
    references CLIENTE (CEDULA);

alter table COMPRAR
  add constraint FK_COMPRAR_REFERENCE_VEHICULO foreign key (NRO_MATRICULA)
    references VEHICULO (NRO_MATRICULA);
```

1. Indices creados
2. Implementación de mecanismos de seguridad y control de acceso

2 — MODELO CONCEPTUAL: ENTIDAD INTERRELACIÓN

2.1 Introducción

En este capítulo describiremos el Modelo Entidad Relación (E/R), mencionado en el capítulo anterior como un modelo de datos y ampliamente utilizado como modelo conceptual en el diseño de bases de datos relacionales. El modelo E/R debido a su capacidad de expresividad y abstracción permite representar la semántica de una situación del mundo real en un diagrama fácil de entender para diseñadores y usuarios. El modelo fue propuesto por Peter Chen en 1976 bajo el enfoque de representar todo objeto del mundo real en términos de entidades y las relaciones que se dan entre estas, considerando que esta es la manera más natural de representar el mundo real. Varios autores han escrito acerca del modelo E/R después de la definición original de Chen, los autores han realizado algunas mejoras y aportes que amplían y extienden los conceptos para facilitar el diseño de la base de datos. El modelo E/R facilita el diseño de la base de datos, ya que permite describirla a un nivel de abstracción capaz de aislar aspectos asociadas a la máquina en donde se almacenará y los usuarios que la consultarán, y hacer relevancia en representar solo la información de la semántica del problema. Para Chen, el modelo se basa en identificar aquellas cosas (o adjetivos) que se identifican claramente para representarlas como entidades y encontrar las vinculaciones o relaciones que se dan entre ellas según el mundo real. A través de los años el modelo ha logrado un gran reconocimiento que le ha permitido convertirse en el modelo conceptual más utilizado y recomendado en el diseño de bases de datos. Si bien, algunos autores como Christopher Date no consideran el modelo E/R como un modelo conceptual, si reconocen la gran utilidad para concebir un modelo parcial de la realidad antes de crear las físicamente las tablas que almacenarán los datos.

2.2 Componente Estático

La propuesta de Chen del modelo E/R considera cuatro conceptos básicos: Entidad, Atributo, Dominio y Relación.

2.2.1 Entidad

Una entidad se define como cualquier cosa u objeto del mundo real que puede ser distinguible y posea existencia propia. La existencia puede ser física o abstracta. Cuando hablamos de existencia física se hace referencia a los objetos reales que ocupan un espacio físico en la realidad, mientras que la existencia abstracta se refiere a aquellos elementos que son intangibles y que se definen conceptualmente. A manera de ejemplos, podemos decir que una persona, un libro o un carro tienen existencia física y un viaje o una receta de cocina existen conceptualmente.
Cada entidad debe describir a sus ejemplares, registros u ocurrencias por medio de un conjunto de características o propiedades que son comunes entre ellos, por ejemplo, una entidad persona describe a todos sus ejemplares a través de las características nombre, identificación y fecha de

nacimiento; un ejemplar de persona podría ser "Carlos Perez" con identificación 109883 y con fecha de nacimiento 12/10/1985. Por otro lado, los ejemplares de una entidad deben poderse distinguir de los demás ejemplares de la misma entidad, por ejemplo, una persona se puede diferenciar de otra a través de los valores que tomen sus propiedades nombre, identificación y fecha de nacimiento.
La representación gráfica de una ***entidad*** en el modelo E/R se hace a través de un rectángulo y en su contenido el nombre de la entidad como se muestra en la siguiente figura.

Figura 2.1:

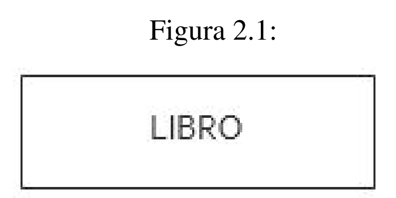


Figura 2.1 Representación de entidad fuerte.

Clases de Entidad

Las entidades pueden clasificarse como *fuertes*, aquellas que poseen ejemplares que existen sin depender de la existencia de otro ejemplar, o *débiles*, en las cuales sus ejemplares dependen de la existencia de otro ejemplar de otra entidad, por ejemplo, consideremos la entidad *LIBRO* en la que se registran los datos de los libros de una biblioteca, sin embargo, para almacenar los datos de las copias que existen de cada libro se necesita la entidad *COPIA,* de esta manera el libro *"Cien años de soledad"* será almacenado como ejemplar de la entidad *LIBRO,* mientras que los datos de todas las copias físicas que existen del libro se almacenarán como ejemplares de la entidad *COPIA.* Si eliminamos un ejemplar de la entidad *LIBRO* se deben eliminar los datos de las copias físicas de éste, por lo tanto, la existencia de los ejemplares de la entidad *COPIA* depende de la existencia de los ejemplares de otra entidad y la debemos clasificar como una entidad débil. La representación gráfica de una ***entidad débil*** en el modelo E/R se hace a través de dos rectángulos (uno dentro de otro) y en su contenido el nombre de la entidad como se muestra en la figura 2.2.

Figura 2.2:

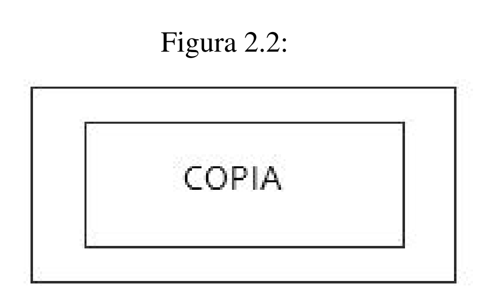


Figura 2.2 Representación de entidad débil.

Identificación de Entidades

Una de las tareas más complejas en el diseño de un modelo E/R es la identificación de las entidades, si bien el concepto de *entidad* puede ser fácil de entender, la aplicación de concepto para determinar si un objeto o cosa del mundo real es una entidad puede ser ambiguo. A continuación se enumeran algunos aspectos que se deben considerar para identificar las entidades en un problema del mundo real:

- En los requisitos o universo del discurso se deben identificar sustantivos que son descritos a través de un conjunto de propiedades.
- Cada entidad identificada debe representar información relevante en el problema del mundo real que se desea modelar.
- Los ejemplares de cada entidad identificada deben poderse diferenciar de los otros ejemplares de la misma entidad.

2.2.2 Atributo

Son las propiedades o características que describen a una entidad o relación. Su existencia depende de la entidad a la que pertenecen y los valores que tomen expresan el estado de un ejemplar.
La representación gráfica de un ***atributo*** en el modelo E/R se hace a través de un ovalo o circulo como se muestra en la Figura 2.3.

Figura 2.3:

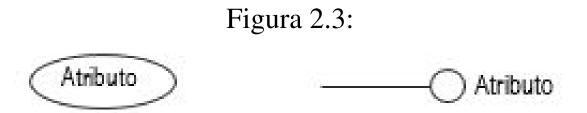


Figura 2.3 Representación de atributos.

Ejemplo 2.1

Para expresar que la entidad *PERSONA* tiene un atributo *Nombre* lo podemos representar como se muestra en la Figura 2.4.

Figura 2.4:

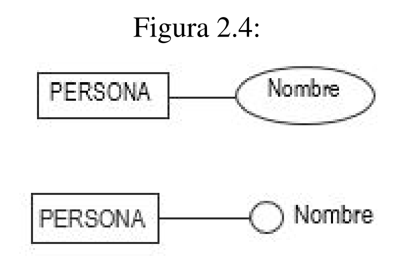


Figura 2.4 Entidad Persona con atributo *Nombre*.

Tipos de atributos

- **Simples o Compuestos**

Los atributos **simples** son atómicos, no poseen ninguna estructura interna y almacenan un único valor. Los atributos **compuestos** son atributos que poseen varios componentes, se usan cuando un grupo de atributos tienen alguna afinidad en su significado en el mundo real o cuando su agrupación en una estructura le da algún significado, por ejemplo, los atributos *Día, Mes* y *Año* los podríamos agrupar en un atributo *Fecha* que posee una estructura más compleja.
La representación gráfica de un atributo ***compuesto*** en el modelo E/R se puede observar en la Figura 2.5.

Figura 2.5:

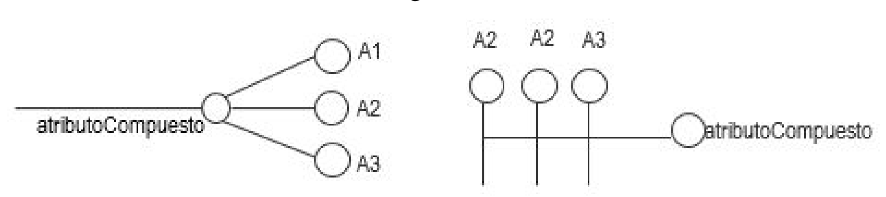


Figura 2.5 Representación *Atributos Compuestos.*

Ejemplo 2.2.

Si consideramos que una persona tiene una característica *fecha de nacimiento* que se compone de *Día, Mes* y *Año*, en el modelo E/R lo podemos modelar como se muestra en la Figura 2.6.

Figura 2.6:

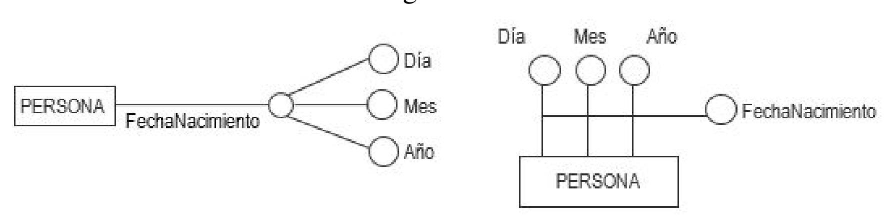


Figura 2.6 Entidad Persona con atributo compuesto *FechaNacimiento.*

- **Derivados o Calculados**Los atributos ***derivados*** o ***calculados*** son aquellos que almacenan un valor calculado a partir de otro atributo. Por ejemplo la edad de una persona, se puede calcular a partir de la fecha de nacimiento.

La representación gráfica de un atributo ***derivado*** en el modelo E/R se puede observar en la Figura 2.7.

Figura 2.7:



Figura 2.7 Representación atributo *Derivado.*

Ejemplo 2.3.

Si consideramos que una persona tiene dos atributos: *FechaNacimiento* y *Edad,* y el atributo *Edad* se calcula a partir del atributo *FechaNacimiento*, el modelo E/R obtenido para esta situación se puede observar en la Figura 2.8.

Figura 2.8:

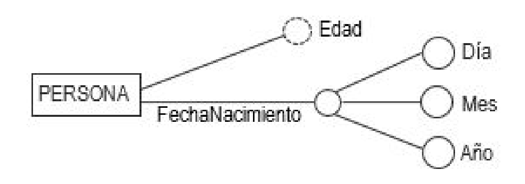


Figura 2.8 Entidad Persona con atributo derivado *Edad.*

- **Univaluado y Multivaluados**

Los atributos ***Univaluados***permiten almacenar un único valor para el ejemplar en esta característica. Por defecto cuando se define un atributo en una entidad, éste es *univaluado.* Los atributos ***multivaluados*** permiten almacenar varios valores para el ejemplar en esta característica. La representación gráfica de un atributo ***multivaluado*** en el modelo E/R se puede observar en la Figura 2.9.

Figura 2.9:

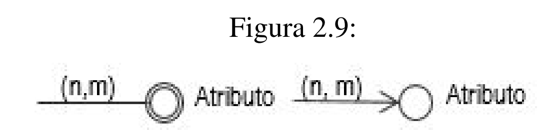


Figura 2.9 Representación atributo *Multivaluado.*

Siendo *n* el número mínimo de valores que se van a almacenar en el atributo y *m* el número máximo.

Ejemplo 2.4.

Si consideramos que de una persona se van a guardar de uno a tres teléfonos, la manera de representarlo en un modelo E/R es emplear un atributo multivaluado como se muestra en la Figura 2.10.

Figura 2.10:

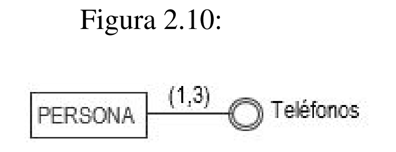


Figura 2.10 Entidad Persona con atributo *Multivaluado.*

- **Opcionales**

Los atributos ***opcionales*** son atributos que por lo general almacenan valores *nulos* o *vacíos.* Se utilizan cuando (i) el valor que se desea almacenar sobre el atributo existe pero no se conoce. (ii) el ejemplar no tiene un valor aplicable en el momento para el atributo, por ejemplo, en un préstamo de un libro, inicialmente no se conoce la fecha de devolución en que un usuario devolverá el libro. La representación gráfica de un atributo ***opcional*** en el modelo E/R se puede observar en la Figura 2.11.

Figura 2.11:

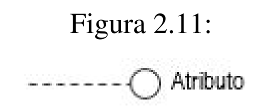


Figura 2.11 Representación de atributo *Opcional.*

Ejemplo 2.5

Si consideramos que algunas personas pueden ser profesionales y por lo tanto, tienen un registro profesional que se debe almacenar, la manera de representar este dato en el modelo E/R es a través de un atributo opcional como se muestra en la Figura 2.12.

Figura 2.12:

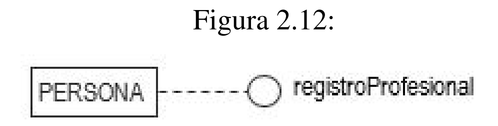


Figura 2.12 Entidad Persona con atributo *Opcional.*

- **Identificador** Un ***Identificador*** de una entidad es un atributo o conjunto de atributos capaces de garantizar una ocurrencia única en dicha entidad, los valores que toman siempre son *no nulos.* Para que un atributo o conjunto de atributos sea considerado *identificador* de una entidad debe cumplir con dos condiciones:

1. No deben existir dos ejemplares de la entidad con el mismo valor en el identificador.
2. No deben tener valores *nulos.*

Por definición todas las entidades fuertes deben tener un ***identificador***. La representación gráfica de un atributo ***identificador*** en el modelo E/R se puede observar en la Figura 2.13.

Figura 2.13:

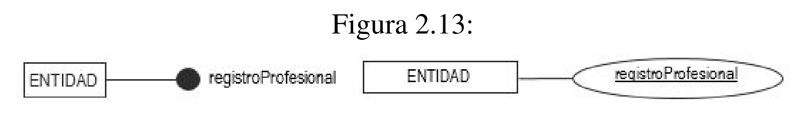


Figura 2.13 Representación atributo *Identificador.*

Ejemplo 2.6

Consideremos que los ejemplares de la entidad *Persona* se identifican a partir del número de célula, la manera de representar este dato en el modelo E/R es a través de un atributo opcional como se muestra en la Figura 2.14.

Figura 2.14:

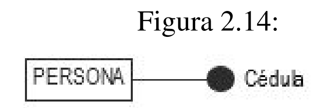


Figura 2.14 Entidad *Persona* con atributo *Identificador Cédula.*

2.2.3 Dominio

Un ***dominio*** es el conjunto de todos los posibles valores que puede tomar un atributo. Los elementos que hacen parte del conjunto de posibles valores comparten una estructura homogénea. Los dominios pueden ser por ***intensión***, aquellos que especifican el tipo de dato, o ***extensión***, aquellos que declaran cada elemento del conjunto de posibles valores. Por definición todos los atributos simples están asociados a un dominio.

Ejemplo 2.7

En la Tabla **2.1** podemos observar 3 dominios declarados para describir los valores validos que pueden tomar los atributos *sexo, edad* y *notaFinal*.

Tabla 2.1 Ejemplos de *Dominio.*

Atributo	Dominio	Descripción Dominio	Tipo
nombre	NOMBRES	cadenas de hasta 30 caracteres alfabéticos	Intensión
sexo	Sexos	F, M, ND	Extensión
edad	Edades	Números enteros entre 0 y 120	Extensión
notaFinal	NotasFinales	números reales entre 0.0 y 5.0	Extensión

No es común usar dentro de un esquema E/R representar gráficamente un *dominio,* sin embargo, en la Figura 2.15 se puede observar cómo expresarlo junto al atributo**.**

Figura 2.15:

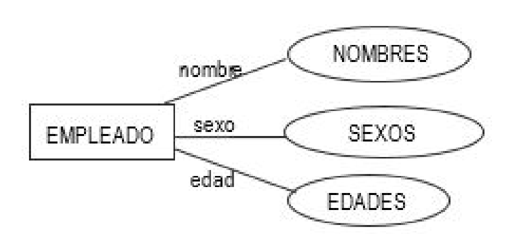


Figura 2.15 Ejemplos de *Dominio*.

2.2.4 Relación

Una relación es una asociación, vínculo o correspondencia que existe entre entidades que se relacionan de alguna manera el mundo real. La representación gráfica de una ***relación*** en el modelo E/R se hace a través de un rombo, el nombre de la relación se pone dentro del rombo y debe ser único. En caso de existir una relación que asocie una entidad débil y esta dependa de la existencia de otra entidad, se debe expresar la relación con un rombo de doble línea. A través de líneas se une el rombo a las entidades que se relacionan como se muestra en la Figura 2.16.

Figura 2.16:

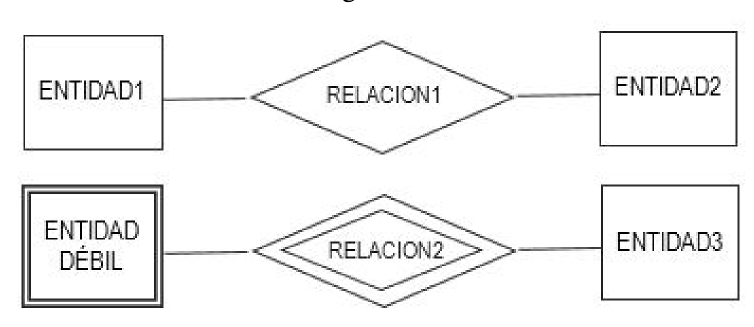


Figura 2.16 Representación de *Relación*.

Ejemplo 2.8

En la Figura 2.17 se establece la relación N*ace* entre las entidades *PERSONA* y *PAÍS.* Como se puede observar la relación *nace* representa la asociación que existe en el mundo real entre las dos entidades.

Figura 2.17:



Figura 2.17 Ejemplo de Relación.

Ejemplo 2.9

En la Figura 2.18 se establecen las relaciones *diseña* y *fabrica* entre las entidades *EMPLEADO* y *PRODUCTO.* Las dos relaciones representan asociaciones que se dan en el mundo real

entre los ejemplares de las entidades.

Figura 2.18:

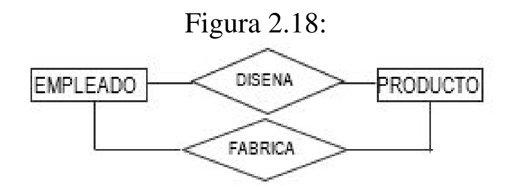


Figura 2.18 Doble relación entre *Empleado* y *Producto*.

> **Nota:** Entre dos entidades puede existir más de una relación que las asocie.

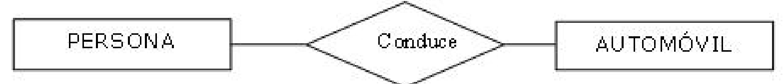
2.3 Semántica de las Interrelaciones

Una interrelación (relación) se considera la representación de una asociación entre los objetos del mundo real, en este caso entre dos entidades. Dentro de ella existen varias consideraciones que propone el modelo conceptual respecto a las interrelaciones a tenerse en cuenta como son: el nombre de la interrelación , el grado de la interrelación, la cardinalidad y el tipo de correspondencia. Estas consideraciones aportan al contenido semántico de las interrelaciones. dentro de la sección 2.3 se ampliaran cada uno de estos conceptos.

2.3.1 Nombre de la Relación

Cada tipo de interrelación tiene un nombre, este permite que se diferencie de las demás interrelaciones que existen en el modelo, esto agrega semántica a la interrelación. En la Figura 2.19, se muestra un ejemplo en el cual participan 2 tipos de entidades: PERSONA y AUTOMOVIL, la interrelación o la asociación entre las dos se llama CONDUCE.

Figura 2.19: Ejemplo de una relación binaria.



2.3.2 Grado de la relación

El grado de una relación (o interrelación), se considera el número de entidades que participan en la relación. En este sentido se puede identificar los siguientes grados que pueden estar presentes en un modelo:

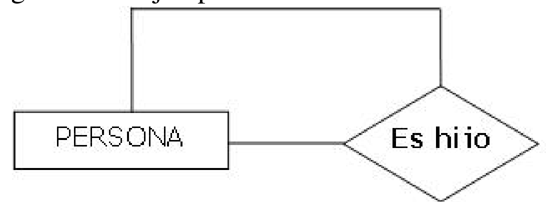
Tabla 2.2. Grados de una relación.

GRADO DE UNA INTERRELACIÓN	NUMERO DE RELACIONES PARTICIPANTES
Unitaria o Reflexiva	1 relación
Binaria	2 relaciones
Ternaria	3 relaciones

Relaciones Unitarias o Reflexivas

En la Figura 2.20, se tiene un ejemplo de lo que se considera una interrelación Unitaria o reflexiva, en la cual participa únicamente una entidad, la entidad persona, indicando que "una PERSONA, es hijo de una PERSONA"

Figura 2.20: Ejemplo de relción reflexiva o Unitaria.



Relaciones Binarias

En la Figura 2.21, se tiene un ejemplo de lo que se considera una interrelación binaria, en la cual participan dos (2) entidades: PERSONA y VIVIENDA, indicando que üna PERSONA habita en una VIVIENDA".

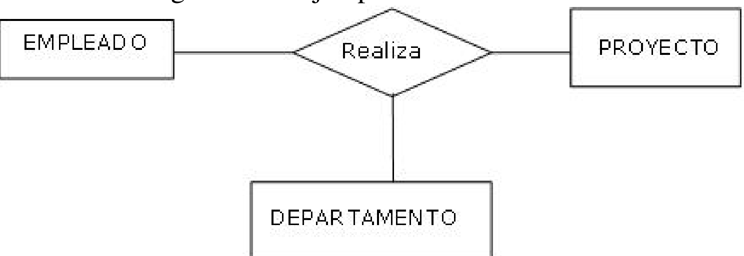
Figura 2.21: Ejemplo de relción Binaria.



Relaciones Ternarias

En la Figura 2.22, se tiene un ejemplo de lo que se considera una interrelación ternaria , en la cual participan tres (3) entidades.

Figura 2.22: Ejemplo de relción Ternaria.



2.3.3 Correspondencia y cardinalidad de las interelaciones

Correspondencia

Se considera como el número máximo de ejemplares de un tipo de entidad que pueden estar asociados con un ejemplar de otro tipo de entidad. Para su representacion, se puede utilizar las etiquetas.

1:N Uno a Muchos
1:1 Uno a Uno
N:N Muchos a Muchos

Cardinalidad

Se considera como el número máximo y mínimo de ocurrencias de un tipo de Entidad que pueden estar interrelacionadas con una ocurrencia del otro y otros tipos de Entidad que participan en el tipo de interrelación. Las cardinalidades se notan con las siguientes etiquetas:

(0,1) Cero a Uno
(1,1) Uno a Uno

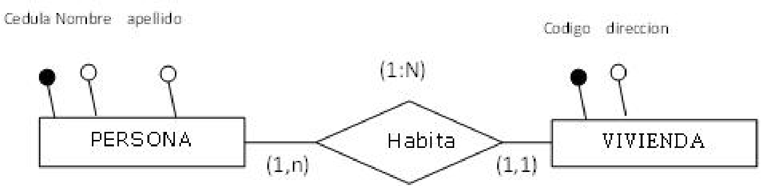
(0,N) Cero a Muchos
(1,N) Uno a Muchos

- Cardinalidad Mínima (el mínimo): Indica el número minimo de relaciones en las que participar acada ejemplar de la entidad. En las cardinalidades minimas los valores que puede tomar es cero (0) o uno (1).
- Cardinalidad máxima (el máximo). Indica el número máximo de relaciones en las que puede aparecer cada ejemplar de la entidad (puede ser uno o muchos)

A continuación se muestran ejemplos con las diferentes cardinalidades y correspondencia:

Ejemplo No. 1

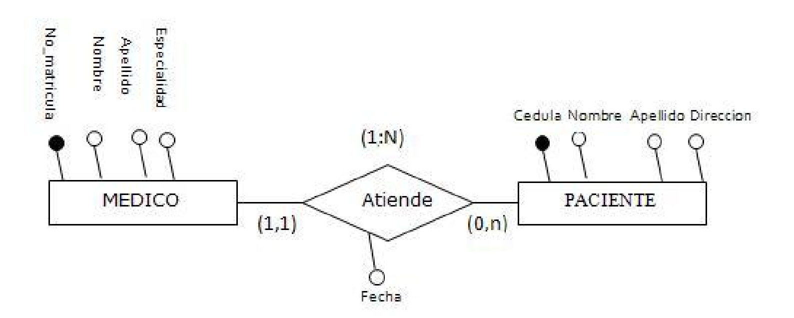
Figura 2.23: Ejemplo No. 1



La Figura 2.23 muestra un ejemplo en el cual dos tipos de entidad PERSONA y VIVIENDA relacionados por medio de la interrelación Habita. La semántica de este modelo indica lo siguiente: "una persona habita minimo una vivienda, máximo una vivienda, y una vivienda es habitada mínimo por una persona máximo por muchas personas." La correspondencia de esta interrelacion es 1:N.

Ejemplo No. 2

Figura 2.24: Ejemplo No. 2



En la Figura 2.24, se muestra un ejemplo en el cual existe una cardinalidad mínima de cero (0), en el cual se relacionan dos tipos de entidad: MEDICO Y PACIENTE. La semantica de este modelo indica lo siguiente: Ün medio puede atender minimo a 0 personas(en caso de que no lleguen los pacientes) y maximo muchos pacientes, y un paciente es atendido por uno y solo un medico.

Ejemplo No. 3

En algunos casos, es importante semánticamente conservar atributos en la interrelación, en el ejemplo anterior, seria importante guardar el periodo en el cual la persona había la vivienda, teniendo en cuenta lo anterior, se le agregaría dos atributos a la interrelación habita, de la siguiente manera, como se ilustra en la Figura 2.25.

Figura 2.25: Ejemplo No. 3

2.4 Mecanismos de Abstracción

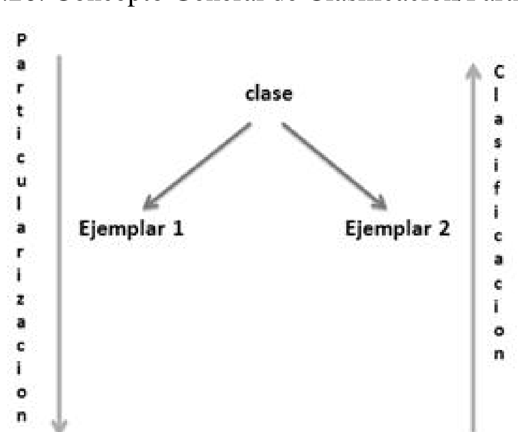
La abstracción es un proceso que permite identificar objetos de la realidad, centrándose en aspectos relevantes para dar soluciones a problemas del mundo real y dejando a un lado aquellos que no se consideran importantes en el entorno estudiado. Por ejemplo, cuando se realiza el proceso de abstracción en un universo del discurso, se identifican diferentes objetos, los cuales interesa centrarse y describir los aspectos que son importantes para el desarrollo y la solución que se va a proponer. Un ejemplo puede ser la identificación del concepto Automóvil, en el cual se debe centrar en las características que son comunes a todos los automóviles y que son relevantes para construir la solución. En un modelo de datos normalmente existen cuatro (4) tipos de abstracción, los cuales ayudan a la estructuración de datos:

- Clasificación
- Agregación
- Generalización
- Asociación

2.4.1 Clasificación

En este mecanismo se hace la abstracción de lo común de un conjunto (general) de ejemplares, identificando que características comparten, a partir de lo cual se crea una categoría a la cual pertenecerán dichos ejemplares. La clasificación es considerada como la acción de abstraer las características comunes a un conjunto de ejemplares para crear una categoría a la cual pertenecen dichos ejemplares. Teoría de conjuntos (Intensión –Extensión). El mecanismo inverso de abstracción es la Particularización. La particularización es pasar de la clase a sus ejemplares. (Pertenencia).
En la Figura 2.26 se muestra en general como se representa el concepto de Clasificación.

Figura 2.26: Concepto General de Clasificación/Particularización.



Por ejemplo, se puede clasificar como vehículos a cualquier elemento que tenga mecanismos de funcionamiento los cuales permitan desplazarse de una posición a otra. En este sentido se puede considerar una clasificación de vehículo como se muestra en la Figura 2.27 en el cual se puede visualizar claramente este mecanismo sobre el concepto de vehículo.

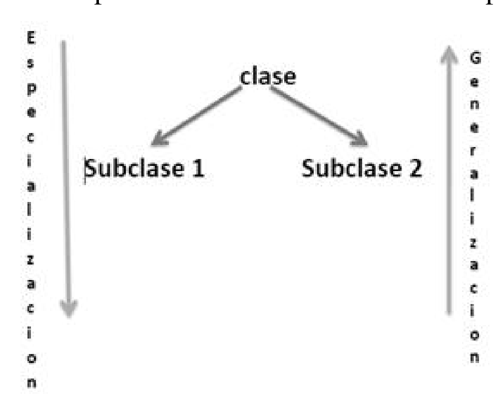
Figura 2.27: Ejemplo del mecanismo de abstración de clasificación/particularización.

En este caso los ejemplares de una clase, presentan características similares, las cuales describen la clase y dichas características toman valores concretos para cada uno de los ejemplares.

2.4.2 Generalización

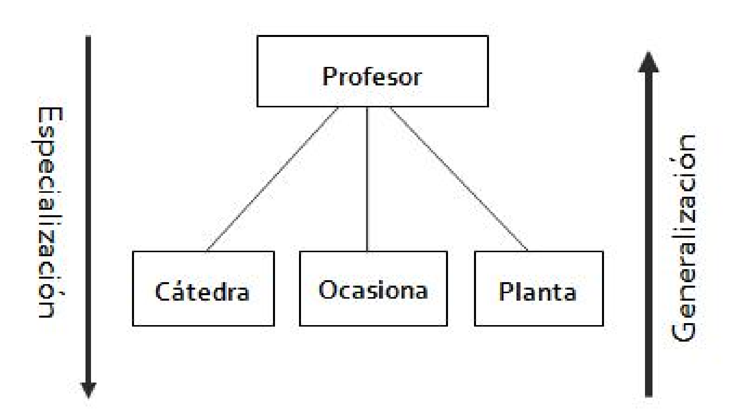
Este mecanismo de abstracción busca identificar las características comunes a varias clases, con el fin de construir una clase general, en la jerarquía que se forma se admite únicamente un solo nivel, se considera una única superclase y las subclases necesarias que dependan directamente de la superclase. El conjunto de ejemplares de una subclase "es un" subconjunto de los ejemplares de la correspondiente superclase. Todo ejemplar de una subclase, es también ejemplar de una superclase. El mecanismo contrario se considera Especialización. En la Figura 2.28 se muestra la forma general del concepto de Generalización.

Figura 2.28: Concepto General de Generalización/Especialización.



En la Figura 2.29, se muestra un ejemplo del mecanismo de abstraccion de Genealización/Especialización.

Figura 2.29:



2.5 Modelo Entidad Relación Extendido

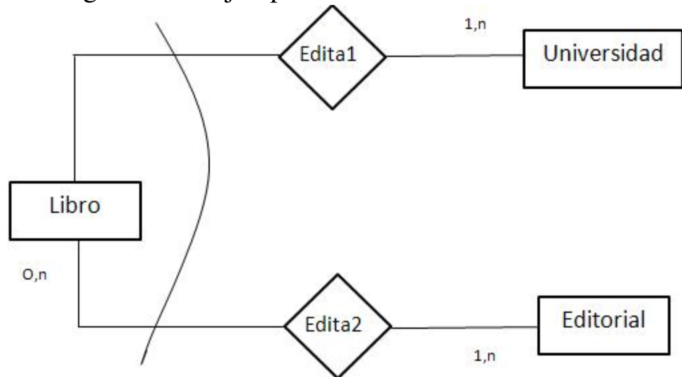
Hasta el momento, se han consideralos los elementos basicos del modelo Entidad/Relacion, sin embargo se crearon extensiones al modelo en los cuales se propone nuevos conceptos semánticos de modelado al modelo ER original, dando lugar al modelo entidad-relación extendido (EER: enhanced Entity-Relationship model). En esta seccion, se tratan los elementos del modelo entidad-Relacion Extendido.

2.5.1 Restricciones sobre interrelaciones

Exclusividad

En una restricción de Exclusividad, dos o más tipos de interrelaciones tienen una restricción de Exclusividad con respecto a un tipo de entidad que participa en ambas interrelaciones si cada ejemplar de dicho tipo de entidad solo puede participar en uno de los tipos de la interrelación a la vez, es decir en el momento en que participa en uno ya podría formar parte del otro.

Figura 2.30: Ejemplo de restricción de Exclusividad.



En el ejemplo anterior, un libro es editado por una universidad o por una editorial, pero no por los dos a la vez.

Exclusión

Un Ejemplar de una entidad A unicamente puede relacionarse con un ejemplar de una entidad B a travez de una interrelación I1 o U2, pero o ambas a la vez. A continuación, se presenta el

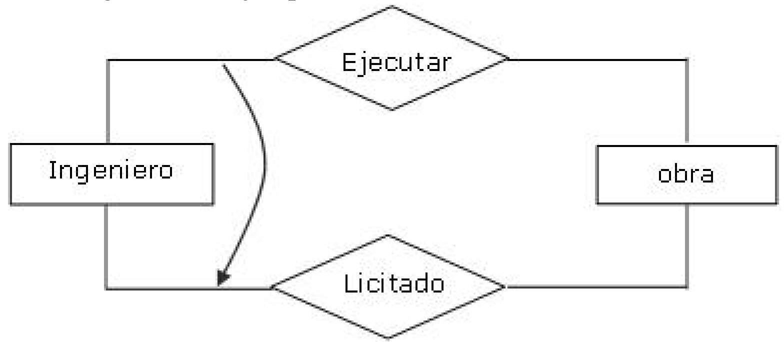
ejemplo de la restricción de exclusón, en la cual Todo ejemplar de profesor que este unido a un ejemplar de curso mediante la interrelación imparte, no podrá estar unido al mismo ejemplar de curso mediante la interrelación recibe.

Figura 2.31: Ejemplo de una Restricción de Exclusión.

Inclusividad

Una restricción de inclusividad se presenta cuando todo ejemplar de una entidad A que participa en una interrelación I2 ha tenido que que participar previamente en la interrelación I1. En el siguiente ejemplo, se muestra una restricción de inclusividad, en la cual un ingeniero para ejecutar una obra, debe haber licitado la misma obra incialmente.

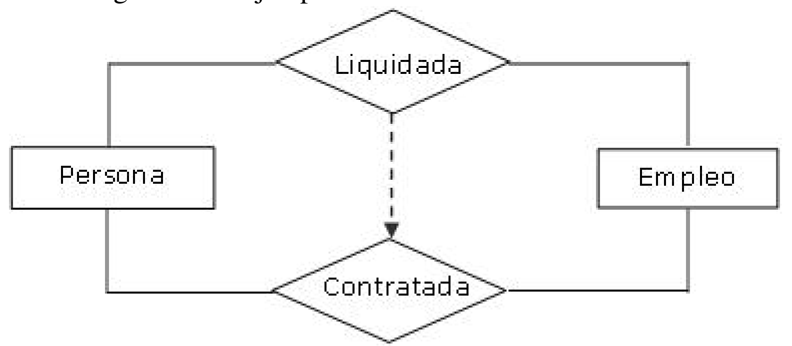
Figura 2.32: Ejemplo de una restrición de Inclusividad.



Inclusión

Se establece una restricción de inclusión, cuando todo ejemplar de una entidad A , para participar en la asociación con un ejemplar de otra entidad B mediante una interrelación, es necesario que ambos ejemplares esten asociados mdiante una segunda interrelación. En la Figura 2.33, se presenta un ejemplo en el cual todo ejemplar de persona que este unido a un ejemplar de empleo, mediante la interrelación liquidada, tiene necesariamente que estar unido al mismo ejemplar de empleo mediante la interrelación contratada.

Figura 2.33: Ejemplo de una restriccion de Inclusión.



2.5.2 Generalización

La generalización se considera como un caso especial de interrelación entre uno o varios tipos de entidad(sub-tipos) y un tipo más general (supertipo, cuyas características son comunes a todos los subtipos). La interrelación que se establece entre los subtipos y el supertipo se considera como “ UN TIPO DE “, Las cardinalidades mínimas y máximas siempre son (1,1) en el supertipo y (0,1) en los subtipos. En una generalización, toda propiedad (atributo, identificadores o participación en tipos de interrelación) del supertipo pasa a ser un atributo de los subtipos, las propiedades comunes a todos los subtipos se asignan al supertipo, mientras que las propiedades especificas se asocian al subtipo al cual pertenecen.La división en subtipos (especialización) puede venir determinada por una condición predefinía (por ejemplo, en función de los valores de un atributo llamado discriminante). La generalización /especialización tiene dos restricciones semánticas asociadas:

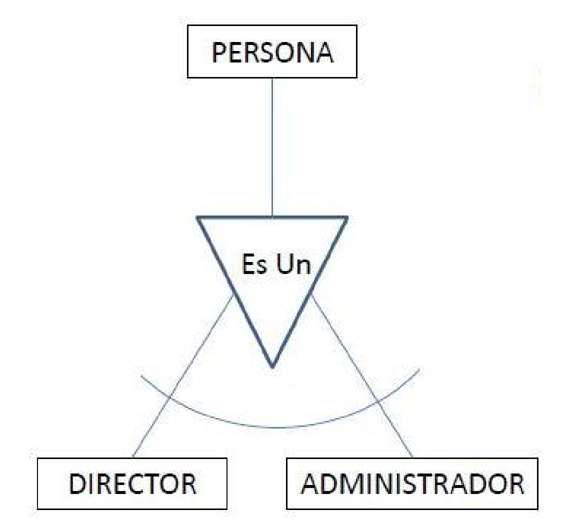
- **Totalidad**: todo ejemplar del supertipo tiene que pertenecer a algún subtipo. El caso contrario se llama parcialidad
- **Parcialidad**: todo ejemplar del supertipo, tiene que pertenecer a algun sub-tipo.
- **Solapamiento**: un mismo ejemplar del supertipo puede pertenecer a más de un subtipo. El caso contrario se llama Exclusividad.
- **Exclusividad:** todo ejemplar del supertipo, tiene que ertenecer a algún subtipo.

Restricciones de la generalización

Parcial exclusiva

A continuacion se presenta una generalización, en la cual participan las entidades persona, director y administrador. Esta restricción se considera parcial debido a que un ejemplar de la entidad persona no es obligatoriamente un director o un administrador, y es exclusiva porque el ejemplar es o director o administrador, no puede ser los dos al tiempo.

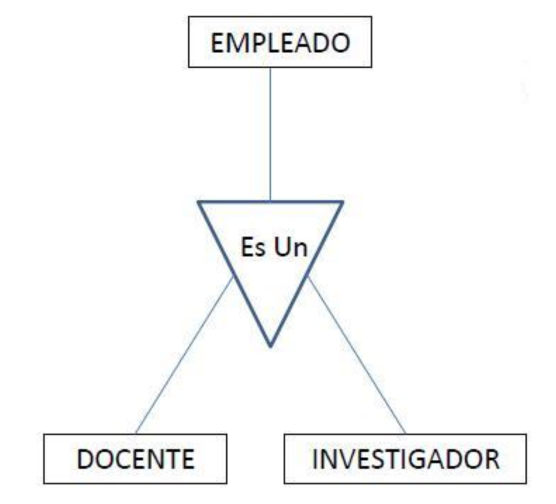
Figura 2.34: Ejemplo de una Jerarquía Parcial Exclusiva.



Parcial solapada

A continuación se presenta una generalización, en la cual participan las entidades empleado, docente e investigador. Esta restricción se considera parcial debido a que un ejemplar de la entidad empleado no es obligatoriamente un director o un administrador, y es solapada porque el ejemplar es o docente puede ser tambien investigador.

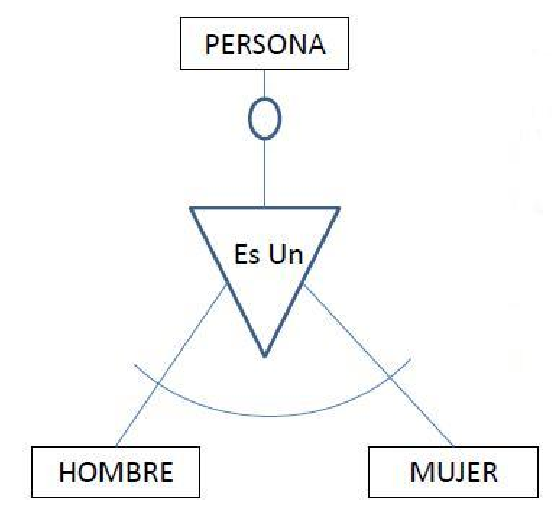
Figura 2.35: Ejemplo de una Jerarquía Parcial Solapada.



Total exclusiva

A continuación se presenta una generalización, en la cual participan las entidades persona, hombre y mujer. Esta restricción se considera total debido a que un ejemplar de la entidad persona debe ser obligatoriamente un hombre o una mujer y exclusiva porque debe ser uno de los dos subtipos, no puede ser los dos a la vez.

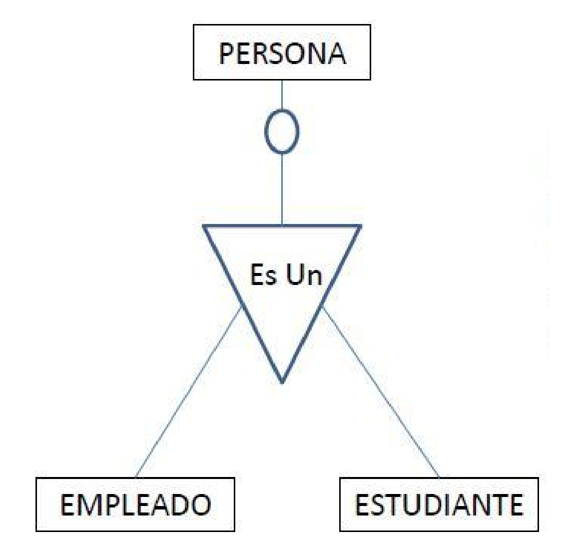
Figura 2.36: Ejemplo de una Jerarquia Total Exclusiva.



Total solapada

A continuacion se presenta una generalización, en la cual participan las entidades persona, empleado y estudiante. Esta restricción se considera total debido a que un ejemplar de la entidad persona debe ser obligatoriamente un empleado o un estudiante y es solapada porque el ejemplar puede ser empleado o administrador a la vez.

Figura 2.37: Ejemplo de una Jerarquia Total Solapada.



2.5.3 Agregación

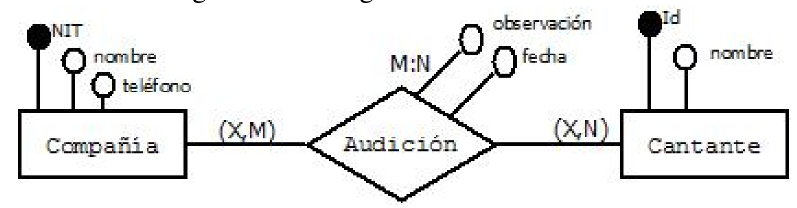
El Modelo Entidad Relación no permite expresar relaciones entre relaciones o entre una relacion y una entidad. Para esta restricción inherente el Modelo Entidad Relación Extendido (MERE) permite crear una **entidad agregada** de nivel superior, la cual combina varias entidades relacionadas mediante una relación. Es de gran utilidad cuando se desea relacionar una relación con otras entidades. La representación gráfica de una ***agregación*** o ***entidad agregada*** se hace a través de un rectángulo que envuelve las entidades y relación que deseamos asociar a través de

una nueva relación a otra entidad.

Figura 2.38: Representación de *Agregación* o *Entidad Agregada.*

A continuación tenemos un esquema E/R que almacena la información de las audiciones que organiza un empresario “Caza talentos” entre futuras estrellas de la música y algunas compañías discográficas.

Figura 2.39: Diagrama Entidad Relación.

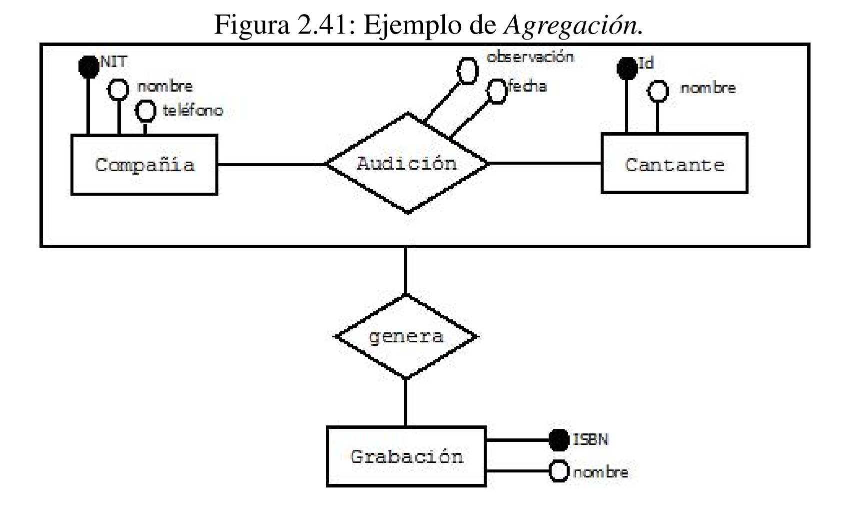


Se sabe que a partir de algunas audiciones se pueden generar grabaciones y las otras en las que los cantantes no tuvieron un buen desempeño no generarán ninguna grabación. Esta situación podría ser erróneamente modelada de la siguiente manera:

Figura 2.40: Solución errónea.

La solución planteada es errónea debido a que una relación ternaria indicaría que todas las audiciones generan una grabación, lo cual no corresponde con la semántica del problema. Una forma fácil de resolver este problema es definir una entidad agregada a partir *Compañía,Cantante* y la relación *Audición* para relacionarla con la entidad *Grabación.*

Figura 2.41: Ejemplo de *Agregación.*



3 — MODELO LÓGICO: RELACIONAL

3.1 INTRODUCCIÓN

La modelación entidad-relación. Como es bien conocido, se modelan situaciones reales que La Normalización es un proceso mediante el cual un esquema de Base de Datos se lleva a un nuevo esquema equivalente pero mejorado en términos de calidad de diseño. La calidad del diseño la medimos con respecto a una serie de reglas que el esquema debe cumplir, dependiendo de las reglas que cumple el esquema diremos que pertenece a cierta Forma Normal. La normalización de datos consiste en la aplicación sucesiva de un conjunto de reglas y técnicas relacionadas con la identificación de las relaciones que existen entre los atributos que conforman las relaciones de una base de datos. Esto, con la intención de mejorar las condiciones generales de la base de datos: no perder información, ser lo menos redundante posible, garantizar la integridad de los datos, etc. El proceso de normalización debe llevar a un esquema desde su estado inicial hasta una forma normal sin modificar las dependencias de los datos y por supuesto, sin perder información. Existen distintas formas normales, unas más restrictivas que otras. En términos generales, podemos concebir el conjunto de formas normales (etapas en el proceso de normalización), como etapas incluyentes de la etapa anterior y cada vez más restrictivas y más eficientes. La siguiente figura, muestra de manera esquemática la relación inclusiva de las formas normales que se basan en dependencias funcionales.

Figura 3.1: Formas Normales Basadas en Dependencias Funcionales.

3.2 TRANSFORMACIÓN DEL MODELO ENTIDAD-RELACIÓN (E-R) A MODELO RELACIONAL

Cuando se ha completado la modelación entidad-relación, es necesario hacer la transformación del modelo para pasarlo a tablas del modelo relacional.Para hacer la transformación,

podemos considerar los casos de relaciones binarias (entre dos entidades) con cardinalidades opcionales y obligatorias del tipo 1:1, 1:N y N:M.

3.2.1 Cardinalidad 1:1

Cuando se tiene una relación con cardinalidad del tipo 1:1 (uno a uno). Se puede presentar uno de 3 casos:

Figura 3.2: Representación de relación.Cuando ambos extremos de la relación son obligatorios con cardinalidades máxima y mínima de 1.

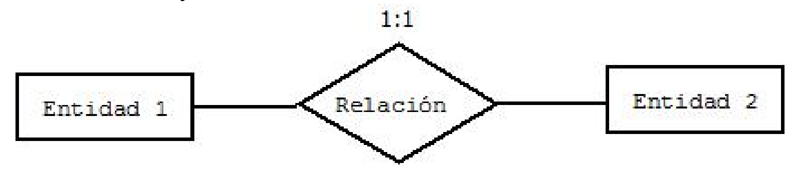
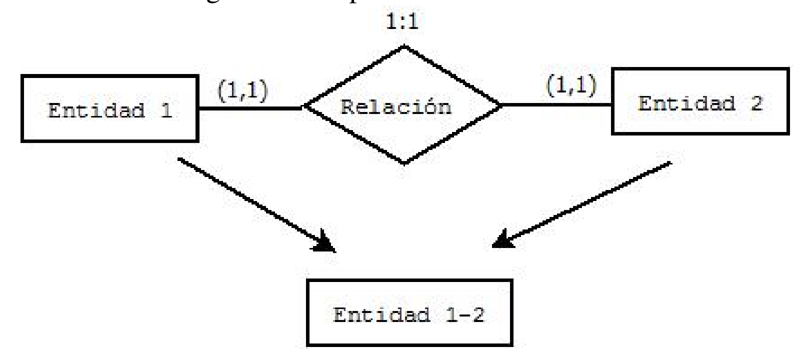
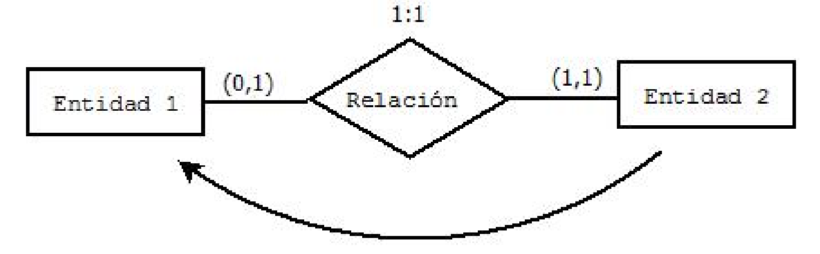


Figura 3.3: Representación de relación.



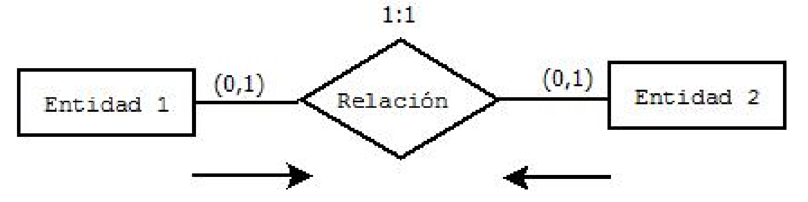
En este caso, se fusionan las dos entidades en una sola relación (o tabla) y su llave primaria será cualquiera de las llaves de la Entidad 1 ó la de la Entidad 2. Como resultado de esto, tendremos únicamente una tabla con todos los atributos.Cuando uno de los extremos de la relación tiene opcionalidad (esto significa que la relación de un lado puede ser (0 ó 1).

Figura 3.4: Representación de relación.



En este caso, la llave primaria del lado de la Entidad 2 pasa a la Entidad 1 como llave foránea y resultando 2 tablas.
Cuando ambos extremos de la relación son opcionales, (o sea que pueden o no existir las conexiones, ser 0 ó 1).

Figura 3.5: Representación de relación.



3.2.2 Cardinalidad 1:N

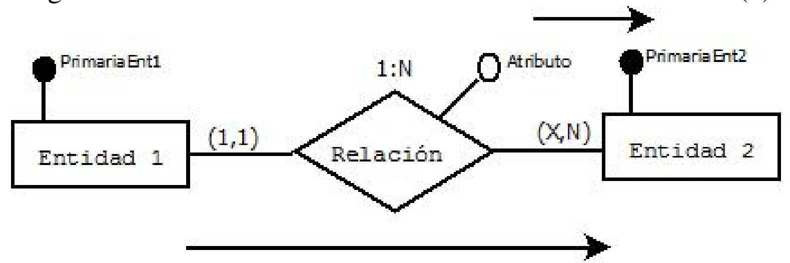
Cuando se presenta una relación con cardinalidad del tipo 1:N (uno a muchos). Hay que revisar la cardinalidad mínima del lado que tiene como máximo 1. Tenemos dos posibles casos:

Figura 3.6: Transofrmación de relación con cardinalidad 1:N.



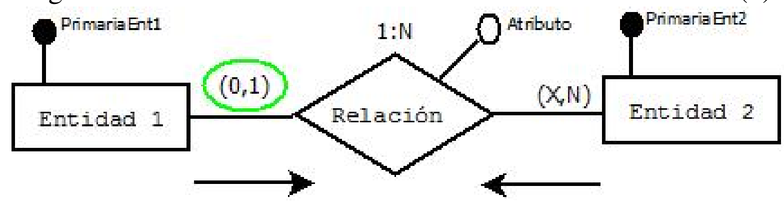
1). Cuando la cardinalidad del lado 1 es (1,1) o sea que la conexión es obligatoria.

Figura 3.7: Transofrmación de relación con cardinalidad 1:N (1).



En este caso, se pasa la llave primaria de la Entidad 1 (la de cardinalidad máxima 1) a la Entidad 2 (la de cardinalidad máxima N) como llave foránea y si la relación contenía algún atributo, éste o éstos se pasan a la Entidad 2 también. Obteniendo como resultado sólo 2 tablas (o relaciones).
2). Cuando la cardinalidad del lado 1 es (0,1) o sea que la conexión es opcional.

Figura 3.8: Transofrmación de relación con cardinalidad 1:N (2).

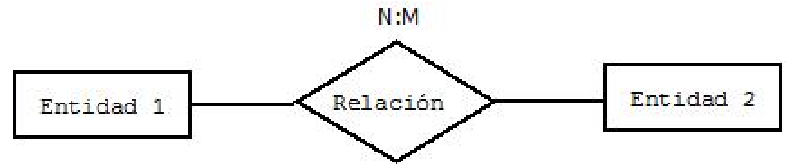


En este caso, la relación entre las entidades, se convierte en una tabla que contendrá sus propios atributos además de las llaves primarias de ambas entidades en los extremos de la relación. Las llaves de las entidades 1 y 2 forman la llave primaria de la relación nueva y también fungirán como llaves foráneas.

3.2.3 Cardinalidad N:M

Cuando se tiene una relación con cardinalidad N:M (o sea, del tipo muchos a muchos entre entidades de los dos lados de la relación)

Figura 3.9: Transofrmación de relación con cardinalidad N:M.



Entonces, deberá de resolverse convirtiendo la relación en una tabla que conserve sus atributos y que además reciba las llaves primarias de las entidades de los lados como llaves foráneas y primarias también.

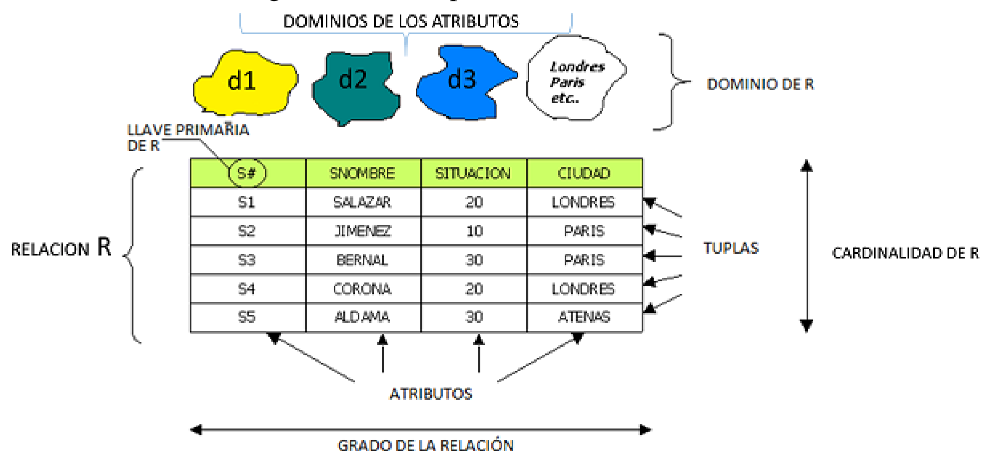
3.3 NORMALIZACIÓN

Conceptos relacionados con la normalización. La normalización es el proceso mediante el cual se transforman estructuras de datos a otras, que además de ser más simples y más deseables, son más fáciles de mantener.
La normalización también puede ser conocida como una serie de reglas de diseño de base de datos que minimizan los problemas de ABC (Altas, Bajas y Cambios).
La normalización busca evitar la creación de tablas redundantes, ineficientes y que además conducen a la generación de errores al momento de manipular la información.
En términos generales, la normalización es un proceso en el que sus resultados parciales (las formas normales) se van construyendo una encima de la anterior. Por tal motivo se dice que se trata de un proceso reversible que se realiza de forma secuencial y consiste en transformar un conjunto de relaciones en otro que sea más deseable por sus características pero equivalente en términos de conservar toda la información.
La normalización se basa en algunos conceptos importantes tales como el de atributo, llave primaria, llave candidato, superllave, dependencia funcional, por lo que se presentan en la siguiente sección de este capítulo.
La normalización es la secuencia por la que un modelo de base de datos relacional es creado y mejorado. La secuencia de etapas implicadas en el proceso de normalización se llama formas normales (FN ó NF).
Básicamente, las formas normales aplicadas durante un proceso de normalización permitirán la creación de una base de datos relacional como un modelo paso a paso de progresión.

3.3.1 Relación

Una relación en el modelo relacional, es un objeto, persona o un hecho sobre el cual, es necesario conservar información. Esto se debe a que su información será consultada posteriormente.
Las relaciones se constituyen de ciertos elementos que pueden verse en la siguiente figura:

Figura 3.10: Componentes de una Relación.



Cada atributo está definido en un cierto dominio. Los dominios de los atributos deben ser simples. El conjunto de dominios de todos los atributos (d1, d2, ..., dn) conforma el dominio de la relación D. La relación tiene una cabecera formada por pares (atributo, dominio). Por ejemplo: el atributo ciudad está definido en el conjunto de nombres de ciudades. Las bases de datos relacionales deben permitir el uso de dominios para los atributos. Sin embargo, un concepto que se asemeja a un dominio es un tipo de dato y es normalmente la manera de subsanar este punto. Una relación tiene también un cuerpo. Éste, se forma por pares del tipo (atributo, valor). Cada tupla contiene valores para los distintos atributos de la relación. Los valores de los atributos pueden ser nulos a menos que el atributo o atributos sean parte de la llave primaria de la relación. Para las llaves foráneas las reglas son menos restrictivas siempre u cuando se consideren todas las implicaciones de asignar valores nulos. Las relaciones cuentan con un atributo o conjunto de atributos que identifican de manera única y mínima cada tupla de la misma. Este atributo o conjunto de atributos son la llave primaria de la relación. El total de tuplas de una relación se denomina cardinalidad de la relación y es una característica muy cambiante. Al total de atributos, se le denomina grado y esta es una característica más permanente a lo largo del tiempo. Un atributo es un elemento que sirve para describir o caracterizar a cada tupla de una relación. Por ejemplo: La relación Alumno:

Tabla 3.1. Relación Alumno

Alumno			
Id	Apellido	Carrera	Semestre
1	Torres	ISC	5
2	Sánchez	ISC	3
3	Benitez	ICI	3
4	Torres	LTI	9

Que puede expresarse como: Alumno (id, Apellido, carrera)
El id es un número identificador que caracteriza de manera única a cada estudiante. El apellido, describe esta característica de cada alumno. Podemos ver que existe un alumno cuyo apellido es Torres que estudia Ingeniería en Sistemas Computacionales (ISC) mientras que otro alumno que estudia Licenciatura en Tecnologías de Información (LTI) se apellida igual, pero no es el mismo alumno.
Por otro lado, el semestre se refiere al grado que actualmente cursa el estudiante, la carrera,

caracteriza el programa de estudios en el que el alumno se encuentra matriculado y el semestre es el grado en que el alumno.
Llave primaria de la relación, como ya se mencionó, es el id, si fuera garantizado que no se repite el apellido, entonces también este atributo podría ser llave primaria (siempre que se considerara atómico o no divisible).

3.3.2 Razones para normalizar

Las bases de datos relacionales se normalizan para:

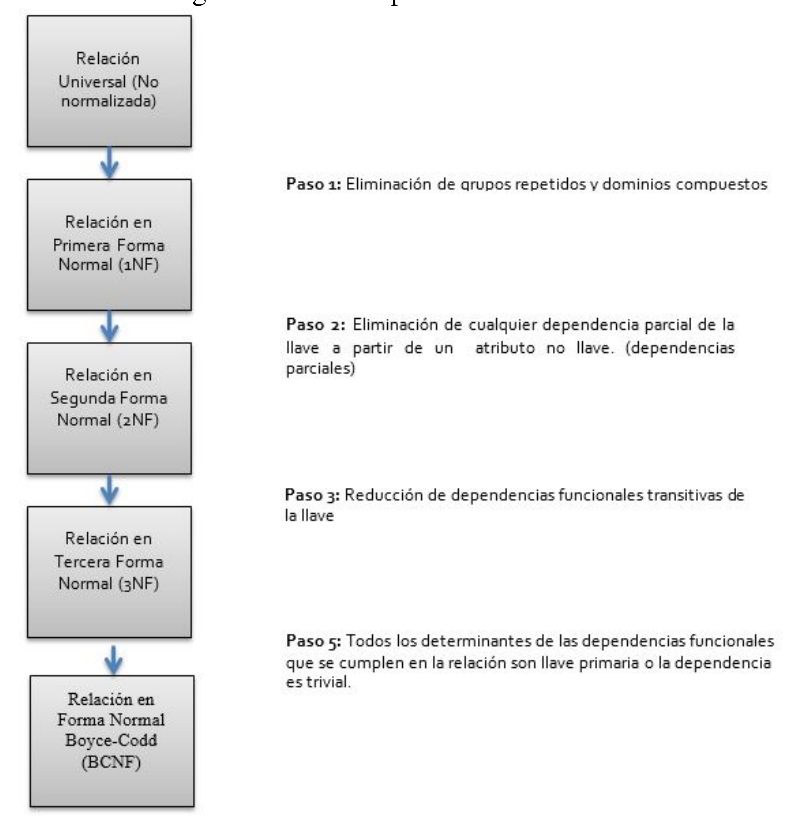
- Evitar la redundancia de los datos.
- Disminuir problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

En el modelo relacional es frecuente llamar tabla a una relación, aunque para que una tabla sea considerada como una relación tiene que cumplir con algunas restricciones:

- Cada tabla debe tener su nombre único.
- No puede haber dos filas iguales. No se permiten los duplicados.
- Todos los datos en una columna deben ser del mismo tipo (pertenecer a un mismo dominio).

La normalización, como se ha comentado anteriormente, es el proceso mediante el cual se transforman estructuras de datos a otras, que además de ser más simples y más deseables, son más fáciles de mantener. Este proceso reversible y ordenado.
El siguiente esquema, muestra el proceso de transformación haciendo uso de los conceptos de llave primaria y dependencia funcional.

Figura 3.11: Pasos para la normalización.



Anomalías de Altas, Bajas y Cambios.

Anomalía: Desperfecto que produce inconsistencia en la información
Consistencia: que exista una sola versión de la información y sea veraz.

Ejemplo de anomalía de altas:

Si tenemos la siguiente tabla (este ejemplo es muy reducido pero suficiente).

Tabla 3.2. Relación Empresa.

Empresa				
No. Departamento	Nombre_Depto	Ubicación	Proyecto	Nombre_Proyecto
1	Contabilidad	Oficina 15	1	Estados de cuenta
2	Contabilidad	Oficina 15	2	Reutilización de recursos
3	Finanzas	Oficina 8	1	Construcción de ala norte
4	Personal	Oficina 3	4	Capacitación del personal

Y deseamos almacenar la información del proyecto 6 con Nombre: Estímulo de productividad (que se asigna al departamento que en el año tuvo la mayor productibilidad). Supongamos que aún no sé a cuál departamento asignar el proyecto. Entonces no puedo almacenar la información del proyecto porque no sé a cuál departamento asignarlo. Además veo que la llave primaria de la tabla se conforma con el número del departamento y el número del proyecto.
Ésta es una anomalía de alta porque si deseo almacenar la información del proyecto 6 no puedo hacerlo sino hasta que tenga la información del departamento al que se lo asignaré. Además si existe un departamento desarrollando 50 proyectos, entonces tendría que repetir la información del mismo 50 veces, (una vez por cada proyecto que le sea asignado).

Ejemplo de anomalía de bajas:

Con la misma información de la relación anterior, supongamos que se desea eliminar el proyecto de capacitación de personal (pues ya terminó y ahora no se tiene presupuesto para mantenerlo activo).
Sería necesario eliminar también la información del departamento de Personal pues supongamos que es el único proyecto que este departamento tiene.
Esta es una anomalía de bajas pues por tratar de eliminar la información del proyecto 4, estaría eliminando también la información del departamento de Personal.

Ejemplo de anomalía de cambios:

Con la información que tenemos en la tabla Empresa, si deseáramos modificar el nombre del proyecto de "Construcción de ala norte" por "Construcción de ala noreste", tendríamos que buscar en toda la tabla por todos los departamentos asociados con este proyecto y cambiar cada uno de ellos.
Como el lector podrá ver, las anomalías de Altas, Bajas y Cambios (ABC) son condiciones indeseables causadas por malos diseños de las estructuras de almacenamiento. La normalización evita caer en este tipo de situaciones indeseables.
Considere la partición de la tabla Empresa en las siguientes estructuras:

Figura 3.12: Partición de la Relación Empresa.

Empresa				
No. Departamento	Nombre_Depto	Ubicación	Proyecto	Nombre_Proyecto
1	Contabilidad	Oficina 15	1	Estados de cuenta
2	Contabilidad	Oficina 15	2	Reutilización de recursos
3	Finanzas	Oficina 8	1	Construcción de ala norte
4	Personal	Oficina 3	4	Capacitación del personal



Departamento		
No. Departamento	Nombre_Depto	Ubicación
1	Contabilidad	Oficina 15
2	Contabilidad	Oficina 15
3	Finanzas	Oficina 8
4	Personal	Oficina 3

Proyecto	
código	val_telefono
1	6354929
2	6382276
1	2262875
6	6316651

Empresa	
código	val_telefono
1	1
1	2
2	2
3	4

Si analizamos la estructura de la imagen anterior, entonces veremos que es factible guardar la información del proyecto 6 aún cuando no lo tenga asignado a un departamento. No es necesario repetir la información del departamento o del proyecto tantas veces como proyectos tenga un departamento o departamentos que se encuentren trabajando en un proyecto. Si necesito eliminar un proyecto, no toco la información del departamento para realizar esta tarea, los cambios tampoco tienen mayor repercusión (si no tocan llaves primarias).

Beneficios de Normalización de Datos

Si se lleva cabo la normalización, se conseguirán los siguientes objetivos de un buen diseño de B. de Datos

1. Eliminar las anomalías de Altas, Bajas y Cambios
2. Se hace posible la expresión de cualquier relación
3. Reducir la necesidad de reestructurar las relaciones a medida que se agregan más datos.

3.3.3 Formas Normales

Teoría de Dependencias Funcionales

Dependencias funcionales.

Son limitaciones del conjunto de relaciones identificadas directamente por el usuario final o implícitas al conjunto dado por las características del mundo real.
Se utilizan para:

- Determinar si una relación R es legal bajo un conjunto dado de dependencias funcionales. Dicho de otra manera, si la relación R satisface a F donde F es el conjunto de dependencias funcionales que aplican a la relación R.
- Para especificar las limitaciones del conjunto de relaciones legales. (que el conjunto de las relaciones satisfaga al conjunto de dependencias funcionales).

El siguiente, es un ejemplo de una relación R (bajo una representación simbólica de valores de atributos.
Entiéndase que el atributo A toma valores del dominio de las $a's$ con un subíndice numérico $(a_1, a_2, \ldots, a_n)$. Lo mismo ocurre con los atributos B, C y D que toman valores de los dominios: $(b_1, b_2, \ldots, b_n)$, $(c_1, c_2, \ldots, c_n)$, $(d_1, d_2, \ldots, d_n)$.

Considerando la relación R:

Tabla 3.3. Relación R.

R			
A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3

Podemos observar que para todo par de tuplas t1 y t2 con valores iguales en su atributo A (esto se expresa como t1[A] = t2[A]), tenemos que su valor en el atributo C coincide, (esto es: t1[C]=t2[C]). En este caso, se puede decir que A→C y no por fuerza ocurre en sentido contrario, en el caso de nuestro ejemplo. C →A no se cumple.

Formalmente, X → Y en R se cumple si y si y solo si ∀ s, t ∈ R, s[X] = t[X] ⇒ s[Y] = t[Y]. Esto es análogo a las funciones: ∀ x1, x2 ∈ X, x1 = x2 ⇒ f(x1) = f(x2), con f: X → Y.

Si sabemos lo anterior, y conocemos la instancia R de la Tabla 3.3, podemos determinar las siguientes posibilidades:

A → C como para ∀ t1, t2 en R, que cumplen t1[A] = t2[A]
se cumple que t1[C] = t2[C].

C → A si consideramos las tuplas 4 y 5 de la relación R:
t4[C] = t5[C] pero t4[A] <> t5[A], por lo tanto, no se cumple que C→A.

AB → D donde AB=A,B =A∪B={A,B}
Debemos buscar si todo par de tuplas iguales en AB son también iguales en D. Analizando la unión de los atributos A ∪ B, podemos observar que no existe un par de tuplas con el mismo valor en la combinación de los atributos AB, por lo tanto, AB es una superllave de R. Al no existir un par de tuplas iguales en AB, se dice que la dependencia es cierta. (no tenemos ningún caso que debata nuestra aseveración).

Como regla general: **Toda superllave de R determina a todas las combinaciones posibles de atributos de la relación R.**

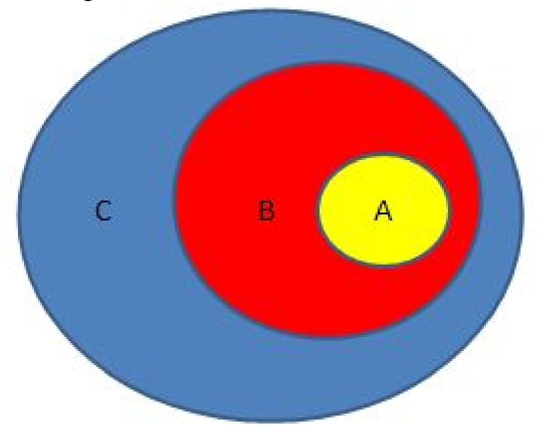
Una superllave de R es cualquier atributo o combinación de atributos que permiten identificar de forma única a cada tupla de la relación. Ninguno de los atributos, por sí solos pueden ser superllave de R porque todos se repiten en R (no identifican de forma única a cada tupla de R).

Sin embargo, si consideramos el atributo A con el atributo B (esto se puede expresar como A∪B | A, B | {A, B} | AB); obtendremos una combinación tal que no se repite para ninguna de las tuplas. Ésta combinación es una superllave de R.

Nótese que si agrego atributos a la combinación, se conserva la característica de que A∪B∪ * sigue siendo una superllave de R. * Significa "cualquier atributo o combinación de atributos". Esto hace que las siguientes combinaciones también sean superllaves de R.
ABC, ABD, ABCD

Dentro del conjunto de las superllaves, existen otras llaves (más restrictivas) que resultan de interés primordial en la normalización de bases de datos, pues las primeras formas normales, dependen de la llave primaria de las relaciones.

Figura 3.13: Llaves de una Relación.



En la Figura 3.13, se puede ver la relación que guardan las llaves de una relación.
El conjunto C corresponde con todas las superllaves que una relación puede tener. El conjunto B, se constituye por todas las llaves candidato de R (todos los atributos o conjuntos de atributos que pueden formar una llave primaria).
Una llave primaria se forma por el atributo o conjunto de atributos que identifican de forma única y mínima a cada tupla de una relación.
Nótese que ahora no es suficiente con identificar de forma única cada tupla, sino que además debe ser un atributo o combinación de atributos mínima. Retomaremos la relación R.

Tabla 3.4. Relación R.

R			
A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3

En la Tabla 3.4. Hemos retomado la relación R para analizar lo siguiente: El atributo A no puede ser llave porque al repetirse, no identifica de forma única a cada tupla de la relación R. Lo mismo ocurre con los atributos B,C y D.Sin embargo, si se combinan el atributo A con el B (AB), de manera combinada tendremos una clave primaria puesto que identifican de forma única y mínima cada tupla de la relación. Decimos que es única porque no se repite para ninguna de las tuplas y es mínima porque si eliminamos cualquiera de sus componentes A ó B, entonces pierde su condición de llave primaria.Si analizamos nuestra relación de ejemplo, podremos notar que la combinación de atributos (AD) también constituye una llave primaria pues se trata de una combinación única y mínima para cada tupla. Incluso la combinación de los atributos (CD) constituyen una llave primaria.Todas las combinaciones existentes que cumplen para ser llave primaria se consideran llaves candidato y cuando se selecciona una de ellas, las demás toman el nombre de llaves alternativas. La combinación de los atributos ABCD es una superllave pues no se repite para ninguna tupla, pero es redundante porque podemos conservar sólo AB y aún es una combinación que no se repite.

Una dependencia funcional es TRIVIAL si todas las tuplas de una relación la cumplen, por ejemplo: A → A, en general, si Y⊆X => X → Y es TRIVIAL (si Y es subconjunto de X) (ejemplo RFC compuesto por **año, mes, día**, **homonimia**, **dígito verificador** siempre determina a **fecha de nacimiento** que se compone de **año, mes día**).

Analizaremos la siguiente relación:

Tabla 3.5. Relación Cliente.

Cliente		
Nombre-cliente	Calle	Ciudad-cliente
Torres	Alamo	Aguascalientes
Ruvalcaba	Benito Juárez	San Luis
Estrada	Rosas	Zacatecas
Soto	Alamo	Aguascalientes
Ibarra	Benito Juárez	San Luis

Se satisface calle → ciudad-cliente pero es posible en el mundo real que ciudades diferentes tengan calles con el mismo nombre, por lo que no conviene incluirla en el conjunto de DF (dependencias funcionales).
Podemos tener dependencias transitivas, si éstas se transmiten de un atributo a otro. Por ejemplo: A → B y B→C entonces A→C a través de B.Tenemos dependencias completas y parciales por ejemplo: AB→CB→C. La dependencia B→C es completa porque no puedo quitar ningún atributo del lado izquierdo de la dependencia (conocido como determinante), sin que se pierda la dependencia.

Pero AB→C es parcial si yo sé que B→C porque puedo omitir a A del lado izquierdo de la dependencia y se sigue cumpliendo.

Para diseñar una B.D relacional, primero se debe generar una lista de todas las dependencias funcionales que existan en el mundo real y para esto utilizamos la teoría de las dependencias funcionales:

- Sea F el conjunto de DF (dependencias funcionales), el conjunto cerrado de F, (F+) contiene todas las dependencias funcionales que F implica.
- Para calcular F+, existen 3 reglas de inferencia y 3 axiomas que se complementan para conocer todas las dependencias existentes en un esquema de bases de datos.

Axiomas o reglas de inferencia (Axiomas de Armstrong)

1. Regla de reflexividad. Si X es un conjunto de atributos, y Y ⊆X, entonces se cumple que X -> Y.
2. Regla de amplificación. Si se cumple que X -> Y, y W es un conjunto de atributos, entonces se cumple que WX -> WY.
3. Regla de transitividad. Si se cumple que X -> Y, y Y->Z, entonces se cumple que X->Z

Para simplificar la aplicación de los axiomas se tiene las siguientes reglas adicionales:

- Regla de unión. Si se cumplen X->Y, y X->Z, entonces se cumple X->YZ.
- Regla de descomposición. Si se cumple que X->YZ, entonces se cumple que X->Y y X->Z
- Regla de seudotransitividad. Si se cumplen X->Y, y WY->Z, entonces se cumple que XW -> Z

Ejemplo 3.1

Si se tiene un esquema R=(A, B, C, G, H, I) y un conjunto de dependencias:

A→B
A→C
CG →H
CG →I
B → H

Se puede obtener

A→H
CG →HI
AG → I

Para comprobar A→H:

Podemos ver que como A→B ^ B→ H por la regla de transitividad, A→H
Para comprobar CG→HI:

Sabemos que CG→H ^ CG→I entonces, por la regla de unión CG→HI

En muchas ocasiones se procura encontrar aquellos atributos que son determinados por un conjunto dado de atributos.
Sea X un conjunto de atributos, al conjunto de todos los atributos determinados funcionalmente por X bajo un conjunto F de dependencias funcionales se le conoce como CONJUNTO CERRADO DE X y se denota por X+

Figura 3.14: Algortimo del cálculo de A+.

A continuación se muestra un algoritmo, escrito en pseudocódigo para calcular A^+

```
Resultado:= A;
while (cambios en resultado) do
      for each dependencia functional del tipo: x → y en F do
            begin
               if y ⊆ resultado then resultado := resultado U y;
            end
      end for
end while
```

Resulta que, en el peor de los casos, este algoritmo puede tardar un tiempo proporcional al cuadrado del tamaño de F.

Ejemplo:

Si se tiene el mismo esquema anterior: R=(A,B,C,G,H,I) y el conjunto de dependencias:

A→B
A→C
CG →H
CG →I
B →H
Calcular AG+

- Empezamos con:
- Resultado = A,G
- Como A→B ∧ A ⊆ Resultado, entonces Resultado=Resultado U B = A, G, B
- Como A→C ∧ A ⊆ Resultado, entonces Resultado=Resultado U C = A, G, B, C

- Como CG $\rightarrow$H $\wedge$ C,G $\subseteq$ Resultado, entonces Resultado = Resultado U H = A, G, B, C, H
- Como CG $\rightarrow$I $\wedge$ C,G $\subseteq$ Resultado, entonces Resultado = Resultado U I = A, G, B, C, H, I

Podríamos continuar según el algoritmo y ver que como B ya está en resultado, entonces H debe incorporarse, sin embargo H ya se encuentra dentro de Resultado. Después de esta primera iteración se desarrolla otra y al no haber cambios ya terminamos. El algoritmo establece que si no hay cambios en Resultado entre 2 iteraciones consecutivas, entonces ya se terminó.Ahora que se tienen las bases teóricas, podemos ver en qué consiste cada una de las formas normales basadas en dependencias funcionales:

Formas normales

1NF. Primera Forma Normal.
Una Relación está en 1NF si todos los dominios de sus atributos son escalares, es decir: los atributos deben ser simples (no compuesto) e indivisibles. Esto descarta:
- Grupos repetitivos - Campos compuestos en los cuáles se pueda tener acceso individual a sus componentes, por ejemplo fechas (dd/mm/aa) o total
2NF. Segunda Forma Normal.
Una relación está en 2NF si - Está en 1NF y - Todos los atributos no-llave, dependen completamente de la llave, dicho de otro modo, si no dependen parcialmente de la llave.
Regla para conseguir el 2NF: Eliminar los atributos no dependientes completamente de la llave y ponerlos en otra tabla estableciendo una relación.

Premisa: Si la llave de una relación no es compuesta, entonces ya está en 2NF

3NF. Tercera Forma Normal.
Una relación está en 3NF si - Está en 2NF y - Ninguno de los atributos no-llave no depende transitivamente de la llavea través de algún otro atributo (o conjunto de atributos) no-llave.

- Esto se verifica si se cumple cualquiera de las 3 siguientes condiciones:
- X$\rightarrow$Y es trivial
- X es superllave del esquema R (nueva)
- Y $\subseteq$ De llave candidato de R (original)
  *Regla para conseguir el 3NF: Eliminar las columnas que no dependan de la llave. Si los atributos no contribuyen a describir la clave, remúevalos a tablas separadas 3NF es suficiente para la mayoría de las aplicaciones, pero si esto no es suficiente, entonces hay más formas normales.***BCNF. Foma Normal de Boyce-Codd.**

Una relación está en BCNF si - Está en 1NF y - Se cumple cualquiera de las 2 siguientes condiciones:

- X$\rightarrow$Y es trivial
- X es superllave del esquema R

EJERCICIOS DE EJEMPLO: FORMAS NORMALES
Primera forma normal (1NF)
Definición formal: Una relación R se encuentra en 1FN si y solo sí por cada renglón columna contiene valores atómicos.
Abreviada como 1FN, se considera que una relación se encuentra en la primera forma normal cuando cumple lo siguiente:

- Las celdas de las tablas poseen valores simples y no se permiten grupos ni arreglos repetidos como valores, es decir, contienen un solo valor por cada celda.
- Todos los ingresos en cualquier columna (atributo) deben ser del mismo tipo.

- Cada columna debe tener un nombre único, el orden de las columnas en la tabla no es importante.
- Dos filas o renglones de una misma tabla no deben ser idénticas, aunque el orden de las filas no es importante. Por lo general la mayoría de las relaciones cumplen con estas características, así que podemos decir que la mayoría de las relaciones se encuentran en la primera forma normal.

Para ejemplificar como se representan gráficamente las relaciones en primera forma normal consideremos la relación alumno cursa materia cuyo diagrama E-R es el siguiente:
Como esta relación maneja valores atómicos, es decir un solo valor por cada uno de los campos que conforman a los atributos de las entidades, ya se encuentra en primera forma normal, gráficamente así representamos a las relaciones en 1FN.
Por ejemplo, si para alguna Base de Datos, su esquema contiene una relación como la siguiente:

Tabla 3.6. Relación Departamento

departamento			
nom_depto	código	fech_creac	teléfonos
Informática	D1	21/03/2014	{6354929,6382276, 2262875}
Marketing	D2	14/05/2012	{6316651, 2775331}
Ventas	D3	11/12/2011	{6382276}
Recursos Humanos	D4	11/02/2013	{2775331}

El esquema completo no se encuentra en 1NF, el atributo *teléfonos* contiene valores no atómicos.
Solución 1: si *r(R)* es la relación que viola 1NF, N el atributo que produce el problema

- Crear una nueva relación *r'(R)* a partir de remover el atributo N de *r'R).*
- Crear una nueva relación *T(K,N')* donde *K* es la clave primaria de *r(R)* de tal manera que las tuplas cumplan:

$$T[N'] \in r[N] \Leftrightarrow t[K] = r[K]$$

- Esta última relación tiene clave primaria *(K, N').*
- Eliminar la relación *r(R)* del esquema de la Base de Datos.

El procedimiento realizado se puede observar en la Figura 3.15

Figura 3.15: Solución 1.

departamento			
nom_depto	código	fech_creac	teléfonos
Informática	D1	21/03/2014	{6354929,6382276, 2262875}
Marketing	D2	14/05/2012	{6316651, 2775331}
Ventas	D3	11/12/2011	{6382276}
Recursos Humanos	D4	11/02/2013	{2775331}



departamento		
nom_depto	código	fech_creac
Informática	D1	21/03/2014
Marketing	D2	14/05/2012
Ventas	D3	11/12/2011
Recursos Humanos	D4	11/02/2013

teléfono	
código	val_telefono
D1	6354929
D1	6382276
D1	2262875
D2	6316651
D2	2775331
D3	6382276
D4	2775331

Solucion 2: Si *r(R)* es una relacion que viola 1NF, *N* el atributo que produce el problema y *K* la clave primaria de .

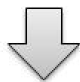
- Crear una nueva relacion $r'(R)$ a partir de remover el atributo N de *r(R)* y agregar un atributo N' de tal manera que las tuplas cumplan:

$$r'[N'] \in r[N] \Leftrightarrow r'[K] = r[K]$$

- La clave primaria de $r'(R)$ es (K, N').
- Eliminar la relacion *r(R)* del esquema de la Base de Datos.

Figura 3.16: Solución 2.

departamento			
nom_depto	código	fech_creac	teléfonos
Informática	D1	21/03/2014	{6354929,6382276, 2262875}
Marketing	D2	14/05/2012	{6316651, 2775331}
Ventas	D3	11/12/2011	{6382276}
Recursos Humanos	D4	11/02/2013	{2775331}



departamento			
nom_depto	código	fech_creac	teléfonos
Informática	D1	21/03/2014	6354929
Informática	D1	21/03/2014	6382276
Informática	D1	21/03/2014	2262875
Marketing	D2	14/05/2012	6316651
Marketing	D2	14/05/2012	2775331
Ventas	D3	11/12/2011	6382276
Recursos Humanos	D4	11/02/2013	2775331

Segunda forma normal (2NF)
Para definir formalmente la segunda forma normal requerimos saber que es una dependencia funcional: Consiste en edificar que atributos dependen de otro(s) atributo(s).
Definición formal: Una relación R está en 2FN si y solo si está en 1FN y los atributos no primos dependen funcionalmente de la llave primaria.Una relación se encuentra en segunda forma normal, cuando cumple con las reglas de la primera forma normal y todos sus atributos que no son claves (llaves) dependen por completo de la clave. De acuerdo con esta definición, cada tabla que tiene un atributo único como clave, está en segunda forma normal.
Dependencia funcional: Consiste en edificar que atributos dependen de otro(s) atributo(s). Sean *X, Y* y *B* conjuntos de atributos de cierta relación.

- Si $X \rightarrow Y$ y para todo $B \subsetneq X$ se cumple que $B \nrightarrow Y$, entonces se dice que Y es totalmente funcionalmente dependiente de *X* (para definir *Y* se necesita *X* completo).
- Si no, se dice que *Y* es parcialmente funcionalmente dependiente de *X*.

Por ejemplo, supongamos que tenemos el siguiente esquema de relación *empleado-proyecto*:

Tabla 3.7. Relación empleado-proyecto

empleado-proyecto					
rut_emp	*cod_proy*	*horas*	*nom_emp*	*nom_proy*	*lugar_proy*

Para indicar que un empleado trabaja cierto número de horas en cierto proyecto, cada proyecto se ubica en un único lugar y tiene un nombre y código. Claramente la única clave

candidata es (*rut, cod_proy*) y podemos decir que:

- *rut_, cod_proy* → *horas* es una dependencia funcional total.
- *rut_, cod_proy* → *nom_emp* es una dependencia funcional parcial ya que cumple *rut*→*nom_emp*.
- *rut_, cod_proy* → *nom_proy, lugar_proy* es una dependencia funcional parcial ya que *cod_proy*→*nom_proy, lugar_proy*.

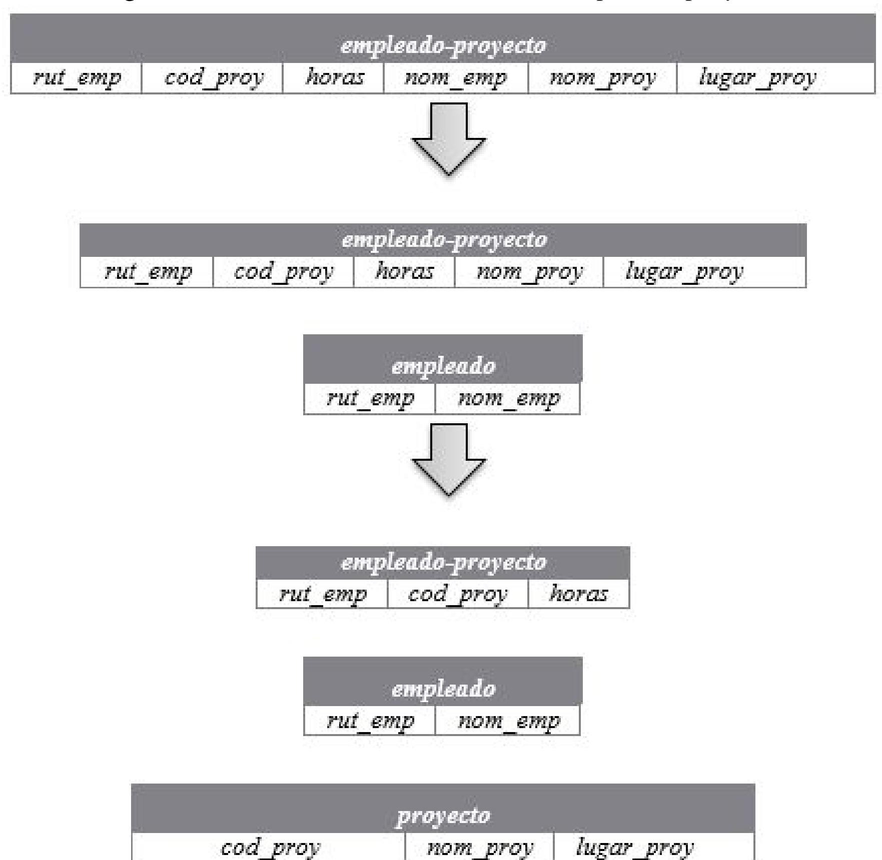
Para entender la idea de 2NF veremos inicialmente un caso particular simple en donde queda claro su aplicación. Supondremos que todas las relaciones tienen una única llave candidata lo que quiere decir que las llaves primarias se encuentran definidas. Un esquema de Base de Datos con la anterior característica se encuentra en 2NF si esta en 1NF y toda relación *r(R)* con llave primaria *K* cumple que para cualquier atributo A que no sea parte de la llave primaria, *A* es totalmente funcionalmente dependiente de *K*.

$K \rightarrow A$ es una dependencia funcional total.

Nótese que si la llave primaria de una relación está compuesta por solo un atributo, la propiedad se cumple siempre.
Un esquema con la relación *empleado-proyecto* de la Tabla 3.7 no se encuentra en 2NF ya que *nom_emp, nom_proy* y *lugar_proy* dependen funcionalmente en forma parcial de la llave primaria.
Solución: Primero convertir en 1NF. Sea *r(R)* una relacion que viola 2NF, y *K* la llave primaria de *r(R).*
Identificar un conjunto de atributos *A* que depende funcionalmente en forma parcial de *K,* y el subconjunto de $K' \subset K$ del que depende funcionalmente en forma total.

- Crear una nueva relación $r'(R')$ que resulta de *r(R)* al eliminar el conjunto de atributos *A*.
- Crear una nueva relación *t(K',A)* de tal manera que las tuplas cumplan: *t[A]=r[A]* $\Leftrightarrow$ *[K']=r[K']*
- La llave primaria de *t(K',A)* es *K'*.
- Eliminar la relación r(R) del esquema.A pesar de que *t(K',A)* esta en 2NF, *r'(R')* posiblemente aun no se encuentra en 2NF, por lo que se debe repetir el proceso hasta que el esquema este en 2NF.

Figura 3.17: Normalización de relación *empleado-proyecto*.



En la anterior formulación las relaciones tenían una única llave candidata y por lo tanto la llave primaria estaba fija. En general este supuesto puede ser no cierto y entonces 2NF cambian en parte su formulación: Un esquema de Base de Datos se encuentra en 2NF si esta en 1NF y en toda relación *r(R)* si un atributo A no es parte de ninguna llave candidata de R, entonces A es totalmente funcionalmente dependiente de todas las llaves candidatas de R. Es simplemente una generalización del caso anterior pero ahora tomando en cuenta todas las llaves candidatas (no solo la primaria).La solución es similar a la anterior salvo que se toman en cuenta en total las llaves candidatas, no solo las llaves primarias. Informalmente:

- Eliminar de la relación inicial los atributos que dependen de forma parcial de parte de una llave candidata.
- Crear una nueva relación con la parte de la llave candidata (como llave primaria) y los atributos asociados.

Por ejemplo, supongamos el siguiente esquema para la relación *lote*:

Tabla 3.8. Relación Lote

lote					
cod_propiedad	*municipio*	*numero_lote*	*área*	*precio*	*impuesto*

Donde las claves candidatas son: cod_propiedad, municipio, numero_lote.
Más la siguiente dependencia funcional: municipio $\rightarrow$ impuesto.
Un esquema con la relación lote no se encuentra en 2NF ya que impuesto depende funcionalmente

en forma parcial de la clave candidata municipio, numero_lote.
Lo solucionamos haciendo:

Figura 3.18: Normalización de la relación *Lote*.

lote					
cod_propiedad	municipio	numero_lote	área	precio	impuesto

lote				
cod_propiedad	cod_municipio	numero_lote	área	precio

Municipio	
cod_municipio	impuesto

Tercera forma normal (3NF)

La regla de la Tercera Forma Normal establece que todas las dependencias parciales se deben eliminar y separar dentro de sus propias tablas. Una dependencia parcial es un término que describe a aquellos datos que no dependen de la llave primaria de la tabla para identificarlos.
Una tabla está normalizada en esta forma si todas las columnas que no son llave son funcionalmente dependientes por completo de la llave primaria y no hay dependencias transitivas. Una dependencia transitiva es aquella en la cual existen columnas que no son llave que dependen de otras columnas que tampoco son llave.
Cuando las tablas están en la Tercera Forma Normal se previenen errores de lógica cuando se insertan o borran registros. Cada columna en una tabla está identificada de manera única por la llave primaria y no debe haber datos repetidos. Esto provee un esquema limpio y elegante, que es fácil de trabajar y expandir. Está basada en el concepto de dependencia funcional transitiva: sean X, Y y Z conjuntos de atributos de cierta relación.
Si $X \rightarrow Y$ y existe un conjunto $Z \neq X$, tal que $X \rightarrow Z$ y $Z \rightarrow Y$ se dice que Y depende funcionalmente en forma transitiva de X a través de Z.
Por ejemplo, supongamos que tenemos el siguiente esquema de relación *empleado - departamento*:

Tabla 3.9. Relación empleado-departamento

empleado-departamento					
nom_emp	sueldo	cod_dept	fecha_ing	nom_dept	fecha_creac

nom_emp llave candidata única (llave primaria). *cod_dept* → *nom_dept, fecha_creac*

En este ejemplo *nom_dept* y *fecha_creac* dependen funcionalmente en forma transitiva de *nom_emp*: *nom_emp* → *cod_dept* y *cod_dept* → *fecha_creac*.
Para entender la idea de 3NF partiremos de un caso particular simple en donde quede claro su aplicación. Supongamos que todas las relaciones tienen una única llave candidata lo que quiere decir que las llaves primarias se encuentran definidas (como en 2NF). Un esquema de Base de Datos se encuentra en 3NF si esta cumple en 2NF y toda la relación *r(R)* con llave primaria *K* cumple que para cualquier atributo *A* que no sea parte de la llave primaria en *R*, *A* no depende funcionalmente en forma transitiva de *K*.

$\nexists$ B tal que K$\rightarrow$B y B$\rightarrow$A

Solución: primero convertir en 2NF. Sea *r(R)* la relacion que viola 3NF y *K* la clave primaria de *r(R):*

- Identificar un conjunto de atributos *A* $(A \nsubseteq K)$ que depende funcionalmente en forma transitiva de *K,* y el conjunto *B* que define la transitividad.
- Crear una nueva relación *r'(R')* que resulta de *r(R)* al eliminar el conjunto de atributos *A*.
- Crear una nueva relación *t(B,A)* tal que: $t[A]=r[A] \Leftrightarrow t[B]=r[B]$.
- La clave primaria de *t(B,A)* es *B*, se debe establecer B como llave foránea en *r'(R')*.
- Eliminar la relación *r'(R')* del esquema.
- *r'(R')* posiblemente aun no se encuentra en 3NF, por lo que se debe repetir el proceso hasta que el esquema este en 3NF.

Figura 3.19: Normalización de la relación *empleado-departamento*.

empleado-departamento					
nom_emp	*sueldo*	*cod_dept*	*fecha_ing*	*nom_dept*	*fecha_creac*

⇓

empleado			
nom_emp	*sueldo*	*cod_dept*	*fecha_ing*

departamento		
cod_dept	*nom_dept*	*fecha_creac*

EJERCICIO ADICIONAL

A continuación se presenta un ejemplo completo:

Tenemos una empresa pública donde los puestos de trabajo están regulados por el Estado, de modo que las condiciones salariales están determinadas por el puesto. Se ha creado el siguiente esquema relacional ***EMPLEADOS (nss, nombre, puesto, salario, emails)*** con **nss** como clave primaria.

Tabla 3.10. Relación empleados

empleados				
nss	*nombre*	*puesto*	*salario*	*emails*
111	*Juan Pérez*	*Jefe de Área*	*3000*	juanp@bd.com; *jefea@bd.com*
222	*José Sánchez*	*Administrativo*	*1500*	*jsanchez@bd.com*
333	*Ana Díaz*	*Administrativo*	*1500*	*adiaz@bd.co,ana@hotmail.com*

Primera forma normal (1FN)

Una tabla está en 1FN si sus atributos contienen valores atómicos. En el ejemplo, podemos ver que el atributo **emails** puede contener más de un valor, por lo que viola 1FN.En general, tenemos una relación **R** con clave primaria **K**. Si un atributo **M** viola la condición de 1FN, tenemos dos opciones.

Solución 1: duplicar los registros con valores repetidos. En general, esta solución pasa por sustituir R por una nueva relación modificada **R'**, en la cual:

- El atributo ***M*** que violaba 1FN se elimina.
- Se incluye un nuevo atributo ***M'*** que solo puede contener valores simples, de modo que si ***R'[M']*** es uno de los valores que teníamos en ***R[M]***, entonces ***R'[K] = R[K]***. En otras palabras, para una tupla con n valores duplicados en ***M***, en la nueva relación habrá n tuplas,

que sólo varían en que cada una de ellas guarda uno de los valores que había en ***M***.

- La clave primaria de ***R'*** es ***(K, M')***, dado que podrá haber valores de ***K*** repetidos, para los valores multivaluados en ***M***. Siguiendo el ejemplo, tendríamos el siguiente esquema para la nueva tabla ***EMPLEADOS'(a)*** con clave primaria ***(nss, email):***

Figura 3.20: Relación *empleados* en 1FN, sollución 1.

empleados				
nss	nombre	puesto	salario	emails
111	Juan Pérez	Jefe de Área	3000	juanp@bd.com; jefea@bd.com
222	José Sánchez	Administrativo	1500	jsanchez@bd.com
333	Ana Díaz	Administrativo	1500	adiaz@bd.co,ana@hotmail.com



empleados' (a)				
nss	nombre	puesto	salario	emails
111	Juan Pérez	Jefe de Área	3000	juanp@bd.com
111	Juan Pérez	Jefe de Área	3000	jefea@bd.com
222	José Sánchez	Administrativo	1500	jsanchez@bd.com
333	Ana Díaz	Administrativo	1500	adiaz@bd.co
333	Ana Díaz	Administrativo	1500	ana@hotmail.com

Solución 2: separar el atributo que viola 1FN en una tabla. En general, esta solución pasa por:

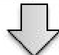
- Sustituir **R** por una nueva relación modificada ***R'*** que no contiene el atributo ***M***.
- Crear una nueva relación ***N(K, M')***, es decir, una relación con una clave ajena ***K*** referenciando ***R'***, junto al atributo ***M'***, que es la variante mono-valuada del atributo ***M***.

La nueva relación ***N*** tiene como clave ***(K, M')***.

Siguiendo el ejemplo, tendríamos el siguiente esquema para la nueva tabla ***EMPLEADOS'(b).*** Y además tendríamos una nueva tabla ***EMAILS*** con clave primaria ***(nss, email):***

Figura 3.21: Relación *empleados* en 1FN, sollución 2.

empleados				
nss	nombre	puesto	salario	emails
111	Juan Pérez	Jefe de Área	3000	juanp@bd.com; jefea@bd.com
222	José Sánchez	Administrativo	1500	jsanchez@bd.com
333	Ana Díaz	Administrativo	1500	adiaz@bd.co,ana@hotmail.com



empleados'(b)			
nss	nombre	puesto	salario
111	Juan Pérez	Jefe de Área	3000
222	José Sánchez	Administrativo	1500
333	Ana Díaz	Administrativo	1500

emails	
nss	emails
111	juanp@bd.com
111	jefea@bd.com
222	jsanchez@bd.com
333	adiaz@bd.co
333	ana@hotmail.com

Segunda forma normal (2FN)

Un esquema está en 2FN si:

- Está en 1FN.

- Todos sus atributos que no son de la clave principal tienen dependencia funcional completa respecto de todas las claves existentes en el esquema. En otras palabras, para determinar cada atributo no clave se necesita la clave primaria completa, no vale con una subclave.

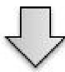
La 2FN se aplica a las relaciones que tienen claves primarias compuestas por dos o más atributos. Si una relación está en 1FN y su clave primaria es simple (tiene un solo atributo), entonces también está en 2FN.Por tanto, de las soluciones anteriores, la tabla ***EMPLEADOS'(b)*** está en 1FN (y la tabla ***EMAILS*** no tiene atributos no clave), por lo que el esquema está en 2FN. Sin embargo, tenemos que examinar las dependencias funcionales de los atributos no clave de ***EMPLEADOS'(a)***. Las dependencias funcionales que tenemos son las siguientes:
nss→ nombre, salario, email
puesto→ salario
Como la clave es ***(nss, email),*** las dependencias de nombre, ***salario*** *y* ***email***son **incompletas**, por lo que la relación no está en 2FN.En general, tendremos que observar los atributos no clave que dependan de parte de la clave.Para solucionar este problema, tenemos que hacer lo siguiente para los grupos de atributos con dependencia incompleta **M**:

- Eliminar de R el atributo M .
- Crear una nueva relación N con el atributo M y la parte de la clave primaria K de la que depende, que
- llamaremos K' .
- La clave primaria de la nueva relación será K' .

Siguiendo el ejemplo anterior, crearíamos una nueva relación con los atributos que tienen dependencia incompleta y al eliminar de la tabla original estos atributos nos quedaría:

Figura 3.22: Relación *empleados' (a)* en 2FN.

empleados' (a)				
nss	nombre	puesto	salario	emails
111	Juan Pérez	Jefe de Área	3000	juanp@bd.com
111	Juan Pérez	Jefe de Área	3000	jefea@bd.com
222	José Sánchez	Administrativo	1500	jsanchez@bd.com
333	Ana Díaz	Administrativo	1500	adiaz@bd.co
333	Ana Díaz	Administrativo	1500	ana@hotmail.com



empleados''			
nss	nombre	puesto	salario
111	Juan Pérez	Jefe de Área	3000
222	José Sánchez	Administrativo	1500
333	Ana Díaz	Administrativo	1500

emails	
nss	emails
111	juanp@bd.com
111	jefea@bd.com
222	jsanchez@bd.com
333	adiaz@bd.co
333	ana@hotmail.com

Como vemos, la solución a la que llegamos es la misma que en la otra opción de solución para el problema de 1FN.
Tercera forma normal (3FN) Una relación está en tercera forma normal si, y sólo si:

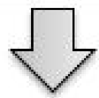
- Está en 2FN

- Y, además, cada atributo que no está incluido en la clave primaria no depende transitivamente de la clave primaria.

Por lo tanto, a partir de un esquema en 2FN, tenemos que buscar dependencias funcionales entre atributos que no estén en la clave. En general, tenemos que buscar dependencias transitivas de la clave, es decir, secuencias de dependencias como la siguiente: $K \rightarrow A$ y $A \rightarrow B$, donde **A** y **B** no pertenecen a la clave. La solución a este tipo de dependencias está en separar en una tabla adicional ***N*** el/los atributos ***B***, y poner como clave primaria de N el atributo que define la transitividad ***A***.
Siguiendo el ejemplo anterior, podemos detectar la siguiente transitividad:
nss→puesto puesto→ salario
Por lo tanto la descomposición sería la siguiente:

Figura 3.23: Relación *empleados"'* en 3FN.

empleados''			
nss	nombre	cod_puesto	salario
111	Juan Pérez	Jefe de Área	3000
222	José Sánchez	Administrativo	1500
333	Ana Díaz	Administrativo	1500



empleados'''			
nss	nombre	cod_puesto	salario
111	Juan Pérez	11	3000
222	José Sánchez	12	1500
333	Ana Díaz	12	1500

puestos	
cod_puesto	puesto
11	Jefe de Área
12	Administrativo

En la nueva tabla ***PUESTOS***, la clave sería el puesto, que también queda como clave ajena referenciando la tabla ***EMPLEADOS***. El resto de las tablas quedan como estaban.

EVALUACION
Ejercicio 1:
Dada la siguiente relación ***PRÉSTAMO_LIBROS (Colegio, Profesor Infantil, Asignatura_Habilidades, Aula, Curso, Libro, Editorial, Fecha_Préstamo)*** que contiene información relativa a los préstamos que realizan las editoriales a los profesores de primaria de los colegios para su evaluación en alguna de las asignaturas/habilidades que imparten.

Tabla 3.11. Relación prestamos_libros

prestamos-libros							
colegio	*profesor*	*Asignatura/habilidades*	*Aula*	*curso*	*libro*	*editorial*	*fechraPrestamo*
C.P. Cervantes	*Juan Pérez*	*Pensamiento Lógico*	*1.A01*	*1°*	*Aprender y Enseñar en Educación Infantil*	*Graó*	*09/09/2013*
C.P. Cervantes	*Juan Pérez*	*Escritura*	*1.A01*	*1°*	*Preescolar Rubio N6*	*Técnicas Rubio*	*05/05/2013*
C.P. Cervantes	*Juan Pérez*	*Pensamiento Numérico*	*1.A01*	*1°*	*Aprender y Enseñar en Educación Infantil*	*Graó*	*06/05/2013*
C.P. Cervantes	*Alicia Garcia*	*Pensamiento Espacial, Temporal y Causal*	*1.B01*	*1°*	*Educacón Infantil N9*	*Prentice Hall*	*06/05/2013*
C.P. Cervantes	*Alicia Garcia*	*Pensamiento Numérico*	*1.B01*	*1°*	*Aprender y Enseñar en Educación Infantil*	*Graó*	*09/09/2012*
C.P. Cervantes	*Andrés Fernández*	*Escritura*	*1.A01*	*2°*	*Aprender y Enseñar en Educación Infantil*	*Graó*	*09/09/2013*
C.P. Cervantes	*Andrés Fernández*	*Inglés*	*1.A01*	*2°*	*Saber Educar: Guía para Padres y Profesores*	*Temas de Hoy*	*05/05/2012*
C.P. Quevedo	*Juan Méndez*	*Pensamiento Numérico*	*1.B01*	*1°*	*Saber Educar: Guía para Padres y Profesores*	*Temas de Hoy*	*18/12/2013*
C.P. Quevedo	*Juan Méndez*	*Pensamiento Numérico*	*1.B01*	*1°*	*Aprender y Enseñar en Educación Infantil*	*Graó*	*06/05/2013*

Se pide responder a los siguientes apartados, considerando las tuplas relación PRÉSTAMO_LIBRO mostradas en la tabla anterior, que a un profesor no se le puede prestar más de un libro de la misma editorial en el mismo día y que a un profesor no se le puede prestar más de una vez un mismo libro:

a) Indicar las dependencias funcionales utilizando las siguientes abreviaturas: Colegio (C), ProfesorInfantil (P), Asignatura_Habilidades (H), Aula (A), Curso (Cu), Libro (L), Editorial (E) y Fecha_Préstamo (F)

b) ¿Cuáles son sus claves? ¿Cuáles son los atributos principales? ¿Y los atributos no principales?

c) ¿En qué forma normal se encuentra la relación? Explicar por qué.

Ejercicio 2: Dada la relación *GASTOS_EMPLEADO(Cod_empleado, Cod_viaje, Destino, Gasto_total)* en la que se cumplen las siguientes dependencias funcionales:

Cod_empleado → Gasto_total Cod_viaje → Destino

Se pide:

a) ¿En qué Forma Normal se encuentra la relación? ¿Por qué?

b) En caso de que la relación no esté en FNBC, ¿cuáles son los problemas que tiene la relación GASTOS_EMPLEADO?

4 — CAPITULO IV - MODELO FÍSICO

4.1 ÁLGEBRA RELACIONAL

El Algebra Relacional es un lenguaje de consulta procedimental, definido a partir de la matemática formal por Edgar Frank Codd en 1972 para conceder el comportamiento dinámico a las bases de datos relacionales. El Algebra Relacional es considerado el precedente más importante que dio origen al primer lenguaje relacional: ALPHA, el cual a su vez, dio los fundamentos del lenguaje SQL, llamado originalmente SEQUEL.
Codd propuso una serie de operaciones basadas en la teoría de conjuntos que generan a partir de una o dos relaciones de la base de datos una relación derivada, asimismo, las relaciones derivadas pueden generar nuevas relaciones derivadas al aplicarse sobre éstas operadores del algebra relacional. Las operaciones definidas pueden ser fundamentales y no fundamentales, siendo las operaciones no fundamentales expresadas a partir de las fundamentales. Por otro lado, las operaciones pueden ser divididas en dos clases, Unarias aquellas que se aplican sobre una relación o Binarias, cuando se aplican sobre dos relaciones.

4.1.1 Operaciones Unarias

Operación de Selección

La operación de selección describe condiciones que deben cumplir los datos de los atributos de una relación R, de tal manera que se crea una nueva relación a partir de las tuplas que cumplen con la(s) condicion(es) definida(s).

Se define como $\boldsymbol{\sigma}_{\ll condiciones \gg}(R)$, siendo $\boldsymbol{\sigma}$ el operador que representa la operación y ***condiciones*** las validaciones que se deben aplicar sobre los atributos de la relación ***R***.

Simples: los operadores simples son los operadores de comparación $=, \neq, <, \leq, >, o \geq$ empleados para validar una condición.

Complejos: los operadores complejos son los operadores lógicos y ($\wedge$), ***o*** ($\vee$) empleados para combinar varias condiciones en una restricción mayor.

Nota: Las condiciones pueden contener combinaciones de operadores simples y complejos.

Ejemplo 4.1.

Consideramos la relación Persona (id, nombre, sexo, edad) que se muestra en la Tabla 4.1. Supongamos que se requiere consultar los datos de las personas que tienen una edad mayor a 18 años, en algebra relacional usando el operador de selección podemos describir esta consulta como

$$\sigma_{edad > 18} (Persona)$$

La relación que resulta de esta consulta se muestra en la Tabla 4.2.

Tabla 4.1. Relación *Persona*

Persona			
id	Nombre	sexo	edad
1	Jhon Masso	M	23
2	Carolina Díaz	F	16
3	Diana Perez	F	15
4	Fabián Ruano	M	29
5	Carlos Sarmiento	F	17

Tabla 4.2. Relación que resulta de $\sigma_{edad > 18}\ (Persona)$

$\sigma_{edad > 18}\ (Persona)$			
id	Nombre	sexo	edad
1	Jhon Masso	M	23
4	Fabián Ruano	M	29

Ejemplo 4.2

El predicado o restricción del operador de Selección puede incluir una combinación de condiciones sobre los atributos de la relación. Por ejemplo, si necesitamos consultar las personas de sexo masculino (M) que tengan menos de 25 años. En algebra relación debemos expresar esta consulta como

$$\sigma_{edad < 25 \wedge sexo = "M"}\ (Persona)$$

El resultado de esta consulta podemos observarlo en la Tabla 4.3.

Tabla 4.3. Relación que resulta de $\sigma_{edad < 25 \wedge sexo = 'M'}\ (Persona)$

$\sigma_{edad < 25 \wedge sexo = "M"}\ (Persona)$			
id	Nombre	sexo	edad
1	Jhon Masso	M	23
5	Carlos Sarmiento	F	17

Ejemplo 4.3

Consideremos la Relación Matricula de la Tabla 4.4. Se requiere consultar todos los estudiantes que tengan una nota mayor igual a 3.0. En algebra relacional podemos expresarlo como

$$\sigma_{NotaFinal \geq 3.0}\ (Matrícula)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.5**.

Tabla 4.4. Relación *Matrícula*

Matrícula	Estudiante	Asignatura	NotaFinal	EdadEstudiante	Docente	EdadDocente
1	Juan Díaz	BD I	3.2	17	María Isabel Vidal	23
2	Catalina Perez	ING. SW	1.5	20	Sandra Roa	29
3	Lucia Jiménez			28		
4	Pedro Ramírez	CÁLCULO	4	19	Fabián Ruano	27
5	Armando Sáenz	BD II	4.2	27	Jhon Masso	23
6	Patricia Márquez	ING. SW	1.0	23	Sandra Roa	29

Tabla 4.5 Relación que resulta de $\sigma_{NotaFinal \geq 3.0}\ (Matrícula)$

Matrícula	Estudiante	Asignatura	NotaFinal	EdadEstudiante	Docente	EdadDocente
1	Juan Díaz	BD I	3.2	17	María Isabel Vidal	23
4	Pedro Ramírez	CÁLCULO	4	19	Fabián Ruano	27
5	Armando Sáenz	BD II	4.2	27	Jhon Masso	23

En la relación que resulta en la Tabla **4.5**, se puede observar que la tupla con los datos de "Lucia Jiménez" de la relación original no se ha tenido en cuenta en el resultado, esto debido a que la comparación realizada en el operador de Selección se ha hecho sobre un atributo que para esta tupla contiene un valor *nulo* o *vacío*.

Nota: Todas las comparaciones que se hagan con un valor nulo se evalúan como falsas.

Ejemplo 4.4

Consideremos la Relación *Matricula* de la Tabla **4.4.** Se requiere consultar todos los estudiantes que tengan una edad mayor a la de su profesor. En algebra relacional podemos expresarlo como:

$$\sigma_{EdadEstudiante > EdadDocente}\ (Matrícula)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.6.**

Tabla 4.6. Relación que resulta de $\sigma_{EdadEstudiante > EdadDocente}\ (Matrícula)$

Matrícula	Estudiante	Asignatura	NotaFinal	EdadEstudiante	Docente	EdadDocente
5	Armando Sáenz	BD II	4.2	27	Jhon Masso	23

Operación de Proyección

La operación Proyección permite definir un subconjunto de campos de una relación que serán mostrados en la consulta resultante. El operador recibe una relación ***R*** de ***n*** campos y el resultado de la proyección es una nueva relación con ***m*** campos, siendo m<=n.

Se define como $\pi_{a_1 \dots a_m}(R)$, siendo π el operador que representa la operación y $a_1 \dots a_m$ los campos que se proyectan de la relación R.

Ejemplo 4.5

Consideremos la relación Entrevista de la Tabla 4.7. Se requiere consultar solo los nombres de los profesores que han practicado pruebas a los estudiantes. En algebra relación podemos expresarlo como

$$\pi_{Profesor}(Prueba)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.8**.

Tabla 4.7. Relación *Prueba*

Prueba			
No. Prueba	Estudiante	Profesor	Evaluación
1	Jhon Masso	Jaime Falla	78
2	Carolina Díaz	Pedro Gómez	89
3	Diana Lagos	Jaime Falla	95
4	Fabián Santos	Lucas Gonzáles	85
5	Carlos Sarmiento	Jaime Falla	90

Tabla 4.8. Relación que resulta de $\pi_{Profesor}(Prueba)$

$\pi_{Profesor}(Prueba)$
Profesor
Jaime Falla
Pedro Gómez
Lucas Gonzáles

En la relación que resulta en la Tabla **4.8**, podemos observar que la operación elimina todas las filas duplicadas, ya que al revisar los datos de la tabla original nos damos cuenta que en el atributo profesor "Jaime Falla" se encuentra en 3 tuplas, sin embargo, la operación por ser una operación de conjuntos solo lo muestra en el resultado una vez.

Nota: En una operación de proyección se eliminan las tuplas duplicadas de la relación resultante.

Composición de Operadores Selección y Proyección

Como ya se ha visto, el resultado de una operación de algebra relacional genera una nueva relación. Por lo tanto, al resultado de una operación se le puede aplicar otra operación. Esto nos permite implementar consultas más complejas.

Ejemplo 4.6

Consideremos la Relación País de la Tabla 4.10. Se requiere consultar solo el nombre de los países que tengan una extensión a 1'500.000 km. En algebra relacional podemos expresarlo como

$$\pi_{NombrePaís}(\sigma_{Extensión > 1'500.000}\ (País))$$

El resultado de esta consulta podemos observarlo en la Tabla **4.10**. En la expresión de algebra relacional podemos ver que la selección de los países con una extensión mayor a 1'500.000 sirve como argumento de la operación de proyección.

Tabla 4.9. Relación *País*

País		
No. País	NombrePaís	Extensión
1	Colombia	2'129.748
2	España	504.645
3	Argentina	2'780 400
4	México	1'964.375
5	Perú	1'285.216

Tabla 4.10. Relación que resulta de $\pi_{NombrePaís}(\sigma_{Extensión > 1'500.000}\ (País))$

$\pi_{NombrePaís}(\sigma_{Extensión > 1'500.000}\ (País))$		
No. País	NombrePaís	Extensión
1	Colombia	2'129.748
3	Argentina	2'780 400
4	México	1'964.375

Operación de Renombramiento

La operación de Renombramiento se aplica sobre una relación R que puede ser el resultado de una operación de algebra relacional. Esta operación es útil ya que las expresiones de algebra relacional no tienen un nombre para referirse a ellas.

Se define como $\rho_X(R)$, siendo ρ el operador que representa la operación y X el nombre que se le asigna a la expresión R.

Si el renombramiento se hace sobre los campos de una relación, el operador se define como $\rho_{X(A1,\ldots,An)}(R)$, siendo ρ el operador que representa la operación, X el nombre que se le asigna a la expresión R y $A1, \ldots, An$ los nuevos nombres de los campos de la relación R.

Ejemplo 4.7

Consideremos la relación que resulta en la Tabla **4.11**. Se desea asignar un nombre a la relación resultante de la expresión de algebra relacional, de tal manera que le nombre sea "*PaisesGrandes*" y sus atributos sean: id, Nombre y Extensión. Esto en algebra relacional podemos expresarlo como

$$\rho_{PaisesGrandes(Id,\ Nombre,\ Extensión)}(\pi_{NombrePaís}(\sigma_{Extensión > 1'500.000}\ (País)))$$

El resultado de esta consulta podemos observarlo en la Tabla **4.11**.

Tabla 4.11. Relación que resulta de $\rho_{PaisesGrandes(Id,\ Nombre,\ Extensión)}(\pi_{NombrePaís}(\sigma_{Extensión > 1'500.000}\ (País)))$

PaisesGrandres		
Id	Nombre	Extensión
1	Colombia	2'129.748
3	Argentina	2'780 400
4	México	1'964.375

4.1.2 Operaciones Binarias

Operación de Unión

La operación Unión permite crear una relación *R* con los elementos de las relaciones *R1* y *R2*. Para poder realizar la operación de unión entre *R1* y *R2*, el par de relaciones deben tener la misma cantidad de atributos y estos ser compatibles entre ellos, es decir, el atributo *i* de *R1* debe tener el mismo, nombre, tipo de dato y dominio del atributo *i* de *R2*

Se define como $R1 \cup R2$, siendo $\cup$ el operador que representa la operación Unión.

Nota: Si la operación $\cup$ tiene como argumentos un par de relaciones que no son compatibles, la operación no será válida.

Ejemplo 4.8

Considere las relaciones *Estudiante* y *Docente* de la Tabla **4.12.** Se requiere obtener los datos de estudiantes y docentes en una nueva relación. En algebra relacional podemos expresar esta consulta como

$$Estudiante \cup Docente$$

El resultado de esta consulta podemos observarlo en la Tabla **4.13**. Las tuplas de ambas relaciones han sido unidas en una nueva relación, por ser una operación de conjuntos, las tuplas repetidas se han eliminado y solo aparecen en la nueva relación una vez.

Tabla 4.12. Relación *Estudiante* y Relación *Docente*

Estudiante	
Cédula	Nombre
1011	Javier Arcos
1012	Carlos Zapata
1013	Jairo Pineda

Docente	
Cédula	Nombre
1111	Sandra Roa
1112	Jhon Masso
1013	Jairo Pineda

Tabla 4.13. Relación que resulta de $Estudiante \cup Docente$

$Estudiante \cup Docentes$	
Cédula	Nombre
1011	Javier Arcos
1012	Carlos Zapata
1013	Jairo Pineda
1111	Sandra Roa
1112	Jhon Masso

Ejemplo 4.9

Considere las relaciones *Estudiante* y *Docente* de la Tabla 4.12. Se desea consultar solos los nombres de estudiantes y docentes. En algebra relacional podemos expresar la consulta como

$$\pi_{nombre}(Estudiante) \cup \pi_{nombre}(Docente)$$

El resultado de esta consulta podemos observarlo en la **Tabla 4.14.**

Tabla 4.14. Relación que resulta de $\pi_{nombre}(Estudiante) \cup \pi_{nombre}(Docente)$

$\pi_{nombre}(Estudiante) \cup \pi_{nombre}(Docente)$
Nombre
Javier Arcos
Carlos Zapata
Jairo Pineda
Sandra Roa
Jhon Masso

Nota: En una operación de Unión $R1$ y $R2$ pueden ser relaciones que resultan de operaciones de algebra relacional.

Operación de Diferencia

La operación Diferencia permite crear una nueva relación con los elementos de la relación *R1* que no están en la relación *R2*. Para que la operación sea válida, las relaciones *R1* y *R2* deben ser compatibles.

Se define como $R1 - R2$, siendo – el operador que representa la operación Diferencia entre dos relaciones.

Ejemplo 4.10

Considere las relaciones *Estudiante* y *Docente* de la Tabla **4.12.** Se desea obtener los Estudiantes que no han sido registrados como docentes. En algebra relacional podemos expresarlo como los estudiantes que no están en la relación , empleando la operación diferencia se representa como

$$Estudiante - Docente$$

El resultado de esta consulta podemos observarlo en la Tabla **4.15**. La tupla de la relación Estudiante que se encuentra duplicada en la relación *Docente* ha sido eliminada.

Tabla 4.15. Relación que resulta de $Estudiante - Docente$

$Estudiante - Docente$	
Cédula	Nombre
1011	Javier Arcos
1012	Carlos Zapata

Operación de Intersección

La operación Intersección es una operación básica de conjuntos que permite crear una nueva relación con los elementos que se encuentran en una relación R1 y una relación R2.

Se define como $R1 \cap R2$, siendo $\cap$ el operador que representa la operación entre dos relaciones, R_1 y R_2.

Considere las relaciones Estudiante y Docente de la Tabla **4.12.** Se desea conocer los estudiantes que también son docentes. En algebra relacional podemos expresarlo como la intersección de las relaciones Estudiante y Docente.

$$Estudiante \cap Docente$$

El resultado de esta consulta podemos observarlo en la Tabla **4.16**.

Tabla 4.16. Relación que resulta de $Estudiante \cap Docente$

$Estudiante \cap Docente$	
Cédula	Nombre
1013	Jairo Pineda

Operación de Producto Cartesiano

La operación de Producto Cartesiano permite crear una nueva relación que concatena cada una de las tuplas de una relación *R1* con cada una de las tuplas de una relación *R2*. De esta manera, la relación resultante tiene *m+n* atributos, correspondiente a los *m* atributos de *R1* y *n*

atributos de *R2*. El número de tuplas de la relación resultante es *MxN,* siendo *M* el número de tuplas de la relación *R1* y *N* el número de tuplas de la relación *R2*.
em>*

Se define como $R1 \times R2$, siendo $\times$ el operador que representa el producto cartesiano entre dos relaciones. Para el caso en que $R1$ y $R2$ tengan atributos con el mismo nombre, a los atributos se les agrega como prefijo el nombre de la relación como se muestra en la **Tabla 4.17**.

Tabla 4.17. Ejemplo Producto Cartesiano

R1	
A	B
1	2
3	4

×

R2		
B	D	E
x	y	q
y	w	z
x	z	w

=

$R1 \times R2$				
A	R1.B	R2.B	D	E
1	2	x	Y	q
1	2	y	W	z
1	2	x	Z	w
3	4	x	Y	q
3	4	y	W	z
3	4	x	Z	w

Cuando se realiza una operación de producto cartesiano entre dos relaciones *R1* y *R2* con atributos con el mismo nombre o que tienen el mismo significado se puede extraer información.

Ejemplo 4.12

Considere las relaciones *Empleado* y *Departamento* de la Tabla **4.18,** ambas relaciones tienen un atributo con el nombre *deptoid*, en la relación *Empleado* el atributo *deptoid* representa el id del departamento al que pertenece un empleado mientras que en la relación *Departamento* el atributo representa el id de cada departamento registrado.

Tabla 4.18. Relación *Empleado* y Relación *Departamento*.

Empleado		
Código	Nombre	deptoid
1120	Javier Santos	11
1121	Carlos Sánchez	12
1122	Jairo Zapata	11
1123	Andrés Perez	11
1124	Fernando Marín	13

Departamento	
deptoid	nomDepartamento
11	Mercadeo
12	Contabilidad
13	Sistemas

Al aplicar producto cartesiano entre las relaciones *Empleado* y *Departamento* obtenemos:

Tabla 4.19. Relación que resulta de Empleado x Departamento.

Empleado x Departamento				
Código	Nombre	Empleado.deptoid	Departamento.deptoid	nomDepartamento
1120	Javier Santos	11	11	Mercadeo
1120	Javier Santos	11	12	Contabilidad
1120	Javier Santos	11	13	Sistemas
1121	Carlos Sánchez	12	11	Mercadeo
1121	Carlos Sánchez	12	12	Contabilidad
1121	Carlos Sánchez	12	13	Sistemas
1122	Jairo Zapata	11	11	Mercadeo
1122	Jairo Zapata	11	12	Contabilidad
1122	Jairo Zapata	11	13	Sistemas
1123	Andrés Perez	11	11	Mercadeo
1123	Andrés Perez	11	12	Contabilidad
1123	Andrés Perez	11	13	Sistemas
1124	Fernando Marín	13	11	Mercadeo
1124	Fernando Marín	13	12	Contabilidad
1124	Fernando Marín	13	13	Sistemas

De la Tabla **4.20** se puede observar que existen algunas filas en las que la columna *Empleado.deptoid* toma los mismos valores que la columna *Departamento.deptoid*. Si consideramos que el atributo *deptoid* en la relación de *Empleado* representa el id del departamento al que pertenece el empleado, se puede concluir que a través del atributo *deptoid* de la relación *Departamento* se accede al nombre del departamento en el que trabaja el empleado si los valores de *deptoid* coinciden en *Empleado* y *Departamento.*

Tabla 4.20. Relación que resulta de *Empleado x Departamento*.

Empleado x Departamento				
Código	Nombre	Empleado.deptoid	Departamento.deptoid	nomDepartamento
1120	**Javier Santos**	**11**	**11**	**Mercadeo**
1120	Javier Santos	11	12	Contabilidad
1120	Javier Santos	11	13	Sistemas
1121	Carlos Sánchez	12	11	Mercadeo
1121	**Carlos Sánchez**	**12**	**12**	**Contabilidad**
1121	Carlos Sánchez	12	13	Sistemas
1122	**Jairo Zapata**	**11**	**11**	**Mercadeo**
1122	Jairo Zapata	11	12	Contabilidad
1122	Jairo Zapata	11	13	Sistemas
1123	**Andrés Perez**	**11**	**11**	**Mercadeo**
1123	Andrés Perez	11	12	Contabilidad
1123	Andrés Perez	11	13	Sistemas
1124	Fernando Marín	13	11	Mercadeo
1124	Fernando Marín	13	12	Contabilidad
1124	**Fernando Marín**	**13**	**13**	**Sistemas**

De esta manera, si se desea consultar los datos de empleados y departamentos a los que pertenecen se deberá aplicar una selección sobre el resultado del producto cartesiano entre las relaciones *Empleado* y *Departamento* para filtrar las filas en las que *Empleado.deptoid* sea igual a *Departamento.deptoid*. En algebra relacional esto es

$$\sigma_{Empleado.deptoid=Departamento.deptoid}(EmpleadoxDepartamento)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.21.**

Tabla 4.21. Relación que resulta de $\sigma_{Empleado.deptoid=Departamento.deptoid}(EmpleadoxDepartamento)$.

$\sigma_{Empleado.deptoid=Departamento.deptoid}(EmpleadoxDepartamento)$				
Código	Nombre	Empleado.deptoid	Departamento.deptoid	nomDepartamento
1120	Javier Santos	11	11	Mercadeo
1121	Carlos Sánchez	12	12	Contabilidad
1122	Jairo Zapata	11	11	Mercadeo
1123	Andrés Perez	11	11	Mercadeo
1124	Fernando Marín	13	13	Sistemas

Operación División

La operación División se aplica sobre dos relaciones R y S cuando se desea realizar una consulta a partir de la expresión "*para todos*".

El resultado de una operación ***división*** corresponde a una nueva relación compuesta por los atributos de ***R*** menos los atributos de ***S***, esto es *(r1,...,rn)*, y las tuplas de ***R*** que tienen correspondencia con la combinación de todas las tuplas de **S**, como se muestra en la Tabla **4.22**.

Tabla 4.22. Explicación de Operación $R \div S$.

R	
R1	S1
a	1
a	2
b	1
b	2
c	2

a y b tienen correspondencia con la combinación de todas las tuplas de S

S
S1
1
2

$R \div S$
R1
a
b

Nota: En una operación de División obligatoriamente los atributos de S deben ser un subconjunto de los atributos de R y el número de atributos de R mayor al número de atributos de S, de lo contrario la operación no será válida.

Ejemplo 4.13

Considere las relaciones *Asignatura* y *Matricula* de las Tablas **4.23** y **4.24**, la relación *Matricula* almacena el id del estudiante y el id de la asignatura matriculada. Se desea consultar los id de los estudiantes que han matriculado todas las asignaturas. En algebra relacional podemos expresarlo como la división entre la relación *Matricula* y una relación que contenga los id de todas las asignaturas. Es importante aclarar que la división se calcula con una relación que

contenga únicamente el atributo id de la asignatura, ya que al hacer la división directamente con la relación Asignatura, el atributo asig_nombre no está contenido en Matricula y no sería válida la operación. Lo anterior lo representamos como

$$Matricula \div \pi_{asig_id}(Asignatura)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.25**. Solo son mostradas la tuplas de la relación *Matricula* que tienen correspondencia con la combinación de todas las tuplas de *Asignatura*.

Tabla 4.23. Relación *Asignatura*

Asigatura	
Asig_id	Asig_nombre
11	Bases de Datos I
12	Ingeniería del Software I
13	Estructuras de Lenguajes
14	Estructuras de Datos

Tabla 4.24. Relación *Matricula*.

Matricula	
Est_id	Asig_id
101	11
101	14
102	11
102	12
102	13
102	14
103	12
103	13
104	11
104	12
104	13
104	14
105	12
106	11
106	12
106	12
106	14

Tabla 4.25. Relación que resulta de $Matricula \div \pi_{asig_id}(Asignatura)$.

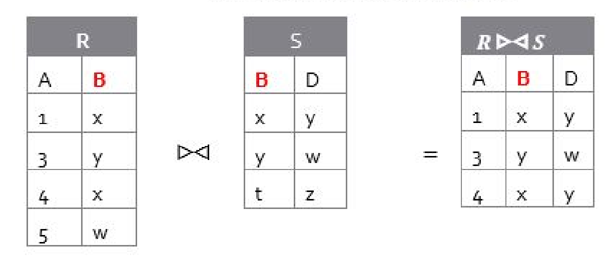
$Matricula \div \pi_{asig_id}(Asignatura)$
Est_id
102
104
106

Operación Combinación Natural (Natural Join)

La operación Combinación Natural se aplica sobre dos relaciones *R* y *S* cuando se desea simplificar consultas que implican una operación de producto cartesiano. El resultado de esta operación es una nueva relación que combina *R* y *S* sobre todos los atributos comunes, esto es, seleccionar en la nueva relación solo las tuplas en donde coincidan los valores de los atributos en común de *R* y *S,* y proyectar los atributos diferentes de *R* y *S* junto a los atributos en común una única vez, a diferencia del producto cartesiano, en donde los atributos en común se proyectan dos veces y se utiliza como prefijo el nombre de la relación para referirse sin ambigüedad a los

atributos en común. El resultado de una Combinación Natural entre R y S se puede observar en la Tabla **4.26.**

Tabla 4.26. Ejemplo Combinación Natural.



R	
A	B
1	x
3	y
4	x
5	w

$\bowtie$

S	
B	D
x	y
y	w
t	z

=

$R \bowtie S$		
A	B	D
1	x	y
3	y	w
4	x	y

Nota: En una operación de **Combinación** Natural entre R y S, si no hay atributos en común entre las dos relaciones el comportamiento de la Reunión Natural será similar al de un Producto Cartesiano.

Ejemplo 4.14

Considere las relaciones *Persona* y *País* de la Tabla **4.27,** ambas relaciones tienen un atributo con el nombre *pai_id,* el atributo *pai_id* en la relación *Persona* representa el id del país de nacimiento de las personas registradas. Se desea consultar los datos de las personas junto al nombre de su país de nacimiento. En algebra relacional podemos expresarlo a través de un producto cartesiano entre *Persona* y *País,* y posteriormente aplicar sobre este resultado una operación de selección para filtrar solo las tuplas en donde hay correspondencia entre los atributos en común:

$$\sigma_{Persona.pai_{id}=País.pai_{id}}(Persona\ x\ País)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.28.**

Tabla 4.27. Relación *Persona* y Relación *País*.

Persona		
Id	Nombre	pai_id
1120	Fabián Ruano	11
1121	Carlos Perez	12
1122	María Vidal	11
1123	Jhon Masso	11
1124	Sandra Roa	13

País	
pai_id	nombrePaís
11	Colombia
12	Argentina
13	México

Tabla 4.28. Relación que resulta de $\sigma_{Persona.pai_{id}=País.pai_{id}}(Persona\ x\ País)$.

$\sigma_{Persona.pai_{id}=País.pai_{id}}(Persona\ x\ País)$				
Código	Nombre	Persona.pai_i	País.pai_id	nombrePaís
1120	Fabián Ruano	11	11	Colombia
1121	Carlos Perez	12	12	Argentina
1122	María Vidal	11	11	Colombia
1123	Jhon Masso	11	11	Colombia
1124	Sandra Roa	13	13	México

Para simplificar la operación $\sigma_{Persona.pai_{id}=País.pai_{id}}(Persona\ x\ País)$ se puede aplicar una combinación natural entre *Persona* y *País*

$$Persona \bowtie País$$

El resultado de esta consulta podemos observarlo en la Tabla **4.29.** En esta consulta se combinan todas las tuplas de *Persona* con las tuplas de *País* en las que coinciden los valores del atributo en común *pai_id*.

Tabla 4.29. Relación que resulta de *Persona* ⋈ *País*.

Persona ⋈ *País*			
Código	Nombre	pai_id	nombrePaís
1120	Fabián Ruano	11	Colombia
1121	Carlos Perez	12	Argentina
1122	María Vidal	11	Colombia
1123	Jhon Masso	11	Colombia
1124	Sandra Roa	13	México

Operación Combinación Externa

Una operación de Combinación Natural solo retorna las tuplas que cumplen con la condición encargada de validar que los atributos en común de ambas relaciones tengan el mismo el valor, mientras que una Combinación Externa considera algunas tuplas que no cumplen la condición. Para dichas tuplas de una relación que no encuentran correspondencia con la otra relación, se completa el resultado con valores nulos.
Existen tres tipos de **Combinación Externa**:

- Combinación Externa a la Izquierda (LEFT OUTER JOIN)
- La Combinación Externa a la Izquierda se aplica sobre dos relaciones R y S, retorna la tuplas de que cumplen con la condición de una Combinación Natural más las tuplas de la relación de la izquierda (R) que no cumplen con la condición.

Se define como R ⟕ S, siendo ⟕ el operador que representa la ***Combinación Externa a la Izquierda*** entre dos relaciones.

Ejemplo 4.15

Considere las relaciones *Docente* y *Universidad* de la Tabla **4.30.** La relación *Docente* tiene el atributo *uni_id* que representa el id de la *Universidad* a la cual pertenece. Se desea consultar los datos de los docentes y en caso de pertenecer a una universidad también se debe mostrar el nombre de la *Universidad.* En algebra relacional podemos expresarlo como una combinación externa a la izquierda entre *Docente* y *Universidad.*

Docente ⟕ *Universidad*

El resultado de esta consulta podemos observarlo en la Tabla **4.31.** La relación resultante contiene las tuplas de *Docente* que tienen correspondencia con las tuplas de *Universidad* más las tuplas de *Docente* que no tienen correspondencia con *Universidad*, en este caso *Carlos Perez* no está asociado a ninguna universidad, por lo tanto los datos con los que se completa el resultado para esta tupla son nulos (vacíos).

Tabla 4.30. Relación *Docente* y Relación *Universidad*.

Docente		
Id	Nombre	uni_id
120	Fabián Ruano	13
1121	Carlos Perez	
1122	María Vidal	11
1123	Jhon Masso	12
1124	Sandra Roa	11

Universidad	
uni_id	nombreUniversidad
11	Universidad del Cauca
12	Universidad de Oriente
13	Universidad Central
14	Universidad Departamental

Tabla 4.31. Relación que resulta de *Docente* ⟕ *Universidad*.

Docente ⟕ Universidad			
Código	Nombre	uni_id	nombreUniversidad
1120	Fabián Ruano	13	Universidad Central
1121	***Carlos Perez***	***null***	***null***
1122	María Vidal	11	Universidad del Cauca
1123	Jhon Masso	12	Universidad de Oriente
1124	Sandra Roa	11	Universidad del Cauca

Combinación Externa a la Derecha (RIGHT OUTER JOIN)

La Combinación Externa a la Derecha se aplica sobre dos relaciones *R* y *S,* retorna la tuplas de que cumplen con la condición de una Combinación Natural más las tuplas de la relación de la derecha (*S)* que no cumplen con la condición.

Se define como *R* ⟖ *S*, siendo ⟖ el operador que representa la ***Combinación Externa a la Derecha*** entre dos relaciones.

Ejemplo 4.16

Considere las relaciones *Docente* y *Universidad* de la Tabla **4.30.** Al aplicar una combinación externa a la derecha entre *Docente* y *Universidad* el resultado incluirá las tuplas de *Universidad* que encuentren correspondencia con *Docente* más las tuplas de *Universidad* (la relación de la derecha) que no encuentren correspondencia con *Docente:*

Docente ⟖ *Universidad*

El resultado de esta consulta podemos observarlo en la Tabla **4.32.**

Tabla 4.32. Relación que resulta de *Docente* ⟖ *Universidad.*

Docente ⟖ Universidad			
Código	Nombre	uni_id	nombreUniversidad
1120	Fabián Ruano	13	Universidad Central
1122	María Vidal	11	Universidad del Cauca
1123	Jhon Masso	12	Universidad de Oriente
1124	Sandra Roa	11	Universidad del Cauca
null	***null***	**14**	***Universidad Departamental***

Combinación Externa Completa (FULL OUTER JOIN)

La Combinación Externa Completa se aplica sobre dos relaciones R y S, retorna la tuplas de que cumplen con la condición de una Combinación Natural más las tuplas de la relación de la izquierda (R) y la relación de la derecha (S) que no cumplen con la condición.

Se define como *R* ⟗ *S*, siendo ⟗ el operador que representa la ***Combinación Externa Completa*** entre dos relaciones.

Ejemplo 4.17

Considere las relaciones *Docente* y *Universidad* de la Tabla **4.30**. Al aplicar una combinación externa completa entre *Docente* y *Universidad* el resultado incluirá las tuplas de Universidad que encuentren correspondencia con *Docente* más las tuplas de *Docente* (la relación de la izquierda) que no encuentren correspondencia con *Universidad* y las tuplas *Universidad* (la relación de la derecha) que no encuentren correspondencia con *Docente:*

$$Docente ⟗ Universidad$$

El resultado de esta consulta podemos observarlo en la Tabla **4.33.**

Tabla 4.33. Relación que resulta de *Docente* ⟗ *Universidad.*

Docente ⟗ Universidad			
Código	Nombre	uni_id	nombreUniversidad
1120	Fabián Ruano	13	Universidad Central
1121	*Carlos Perez*	*null*	*null*
1122	María Vidal	11	Universidad del Cauca
1123	Jhon Masso	12	Universidad de Oriente
1124	Sandra Roa	11	Universidad del Cauca
null	*null*	14	*Universidad Departamental*

4.1.3 Operaciones de Agregación

Las operaciones de agregación son empleadas para obtener resúmenes de los datos de una relación. El resultado de una operación de agregación retorna un único valor y para su cálculo no considera los valores nulos.

Se define como $\Omega_{Función\ (Atributo)}(R)$, siendo Ω el operador que representa el uso de una función de agregación, *Función* la operación de agregación que se desea calcular y *Atributo* la columna de la relación *R* sobre la que se calcula la operación.

Existen cinco tipos de funciones de agregación: COUNT, AVG, MAX, MIN y SUM.

COUNT:

Retorna la cantidad de valores en el atributo indicado sin considerar valores nulos.

Ejemplo 4.18

Considere la relación *Empleado* de la Tabla **4.34.** Se desea calcular la cantidad de salarios registrados. En algebra relacional podemos expresarlo como:

$$\Omega_{COUNT(Salario)}(Empleado)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.35**. La consulta retorna la cantidad de valores no nulos registrados en el atributo *Salario* de la relación *Empleado*.

Tabla 4.34. Relación Empleado.

Empleado			
Id	Nombre	Salario	Departamento
1120	Carlos Vásquez	1'500.000	Sistemas
1121	Julián Perez		Sistemas
1122	Daniela Calle	800.000	
1123	Catalina Prado	2'500.000	Contabilidad
1124	Martin Domínguez	1'400.000	Mercadeo

Tabla 4.35. Resultado de aplicar $\Omega_{COUNT(Salario)}(Empleado)$.

$\Omega_{COUNT(Salario)}(Empleado)$
4

SUM

Retorna la suma de los valores en el atributo indicado sin considerar valores nulos.
Ejemplo 4.19
Considere la relación *Empleado* de la Tabla **4.33.** Se desea calcular el valor total de los salarios registrados. En algebra relacional podemos expresarlo como:

$$\Omega_{SUM(Salario)}(Empleado)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.36**. La consulta retorna la suma de todos los valores no nulos registrados en el atributo *Salario* de la relación *Empleado*.

Tabla 4.36. Resultado de aplicar $\Omega_{SUM(Salario)}(Empleado)$.



$\Omega_{SUM(Salario)}(Empleado)$
6'200.000

AVG

Retorna la media de los valores en el atributo indicado sin considerar valores nulos.
Ejemplo 4.18
Considere la relación *Empleado* de la Tabla **4.34.** Se desea calcular el promedio de los salarios registrados. En algebra relacional podemos expresarlo como:

$$\Omega_{AVG(Salario)}(Empleado)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.37**. La consulta retorna el promedio de todos los valores no nulos registrados en el atributo *Salario* de la relación *Empleado*.

Tabla 4.37. Resultado de aplicar $\Omega_{AVG(Salario)}(Empleado)$.



$\Omega_{AVG(Salario)}(Empleado)$
1'550.000

MIN

Retorna el dato más pequeño de los valores en el atributo indicado sin considerar valores nulos.

Ejemplo 4.21

Considere la relación *Empleado* de la Tabla **4.34.** Se desea calcular el menor salario registrado. En algebra relacional podemos expresarlo como:

$$\Omega_{MIN(Salario)}(Empleado)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.38**. La consulta retorna el dato más pequeño de los valores no nulos registrados en el atributo *Salario* de la relación *Empleado*.

Tabla 4.38. Resultado de aplicar $\Omega_{MIN(Salario)}(Empleado)$.



$\Omega_{MIN(Salario)}(Empleado)$
800.000

MAX

Retorna el dato más grande de los valores en el atributo indicado sin considerar valores nulos.

Ejemplo 4.22

Considere la relación *Empleado* de la Tabla **4.34.** Se desea calcular el mayor salario registrado. En algebra relacional podemos expresarlo como:

$$\Omega_{MAX(Salario)}(Empleado)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.39**. La consulta retorna el dato más grande de los valores no nulos registrados en el atributo *Salario* de la relación *Empleado*.

Tabla 4.39. Resultado de aplicar $\Omega_{MAX(Salario)}(Empleado)$.



$\Omega_{MAX(Salario)}(Empleado)$
800.000

Restricción DISTINCT sobre operaciones de agregación

La restricción ***Distinct*** se agrega sobre las operaciones SUM, COUNT o AVG cuando se desea que el cálculo de la operación se realice solo con los valores distintos del atributo indicado.

Ejemplo 4.23

Considere la relación *Empleado* de la Tabla **4.33.** Se desea calcular la cantidad de departamentos distintos registrados. En algebra relacional podemos expresarlo como:

$$\Omega_{COUNT(DISTINCT\ departamento)}(Empleado)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.40**. La consulta retorna la cantidad de valores no nulos distintos registrados en el atributo *Salario* de la relación *Empleado*.

Tabla 4.40. Resultado de aplicar $\Omega_{COUNT(DISTINCT\ Salario)}(Empleado)$.

$\Omega_{COUNT(DISTINCT\ departamento)}(Empleado)$
3

COUNT (*)

La operación de agregación ***COUNT***puede tomar comoargumento un * en lugar de un atributo, esto indica que se calculará el conteo de todas las tuplas de la relación.

Ejemplo 4.24

Considere la relación *Empleado* de la Tabla **4.33.** Se desea calcular la cantidad de empleados registrados. En algebra relacional podemos expresarlo como:

$$\Omega_{COUNT(*)}(Empleado)$$

El resultado de esta consulta podemos observarlo en la Tabla **4.41**. La consulta retorna la cantidad tuplasde la relación *Empleado*.

Tabla 4.41. Resultado de aplicar $\Omega_{COUNT(*)}(Empleado)$.



$\Omega_{COUNT(*)}(Empleado)$
5

4.2 STRUCTURED QUERY LANGUAGE (SQL) - DATA DEFINITION LANGUAGE (DDL)

El DDL es la parte del lenguaje SQL que realiza la función de definición de datos del SGBD.Fundamentalmente se encarga de la creación, modificación y eliminación de los objetos de la base de datos, Cada usuario de una base de datos posee un esqema. El esquema normalmente tiene el mismo nombre que el usuario y sirve para almacenar los objetos de esquema, es decir los objetos que posee el usuario. Estos objetos pueden ser: tablas, vistas, índices y otras objetos relacionados con la definición de la base de datos. Hay que ener en cuenta que ninguna instrucción DDL puede ser anulada por una instrucción ROLLBACK, por lo que hay que tener recausión a la hora de utilizarlas, Es decir, las instrucciones DDL generan acciones que no se pueden deshacer (únicamente en caso de que se disponga de alguna copia de seguridad).
A continuación, se presentan las sentencias DDL
CREATE: utilizado para crear nuevas tablas, campos, indices y vistas. **DROP**: Empleado para eliminar tablas e índices. **ALTER**: Empleado para modificar tablas **ROLLBACK:** Es utilizada para devolver la base de datos a un estado anterior.

4.2.1 Tipos de Datos Estandar

Para la creación tablas, se hace neceario conocer los tipos de atos, a continuación se presentan los tipos de datos de SQL estándar.

Tabla 4.42. Tipos de datos estándar en SQL

DESCRIPCIÓN	TIPOS ESTÁNDAR SQL
Texto de anchura fija	CHARACTER(n), CHAR (n)
Texto de anchura variable	CHARACTER, VARYING(n), CHAR VARYING(n)
Texto de anchura fija para caracteres nacionales.	NATIONAL CHARACTER, VARYING(n), NATIONAL CHAR, VARYING(n), NCHAR.
Enteros	INGEGER, INT, SMALLINT
Decimal de coma variable	FLOAT (b), DOUBLE, DOUBLE PRECISSION, REAL, NUMERIC(m,d), DECIMAL(m,d)
Decimal de coma fija	NUMERIC(m,d), DECIMAL (m,d)
Fechas	DATE

4.2.2 Sentencias de Creación

Creación de una tabla

En general, la sentencia **CREATE**, es utilizada para crear tablas. A coninuación se presenta la forma general de la creación de una tabla.

```
create Table nombre_tabla
(
nombre_campo_1 tipo_1
nombre_campo_2 tipo_2
nombre_campo_n tipo_n
)
```

por ejemplo, si se necesita crear la tabla PEDIDOS, se realizara de la siguiente manera:

```
Create Table pedidos
(
id_pedido INT(4),
id_cliente INT(4) ,
id_articulo INT(4),
fecha DATE,
cantidad INT(4),
total INT(4)
)
```

Restriciones

1. **Restriccion de Obiligatoriedad.** La restricción **NOT NULL** permite definir si el campo aceptará o no valores nulos. En este caso, todas las columnas de una tabla que tengan la restricción Not Null obliga que tengan un valor para ser almacenado. En el ejemplo anterior, en el cual se creo una tabla llamada pedidos, se le puede inclur a sus columnas la restricción Not Null.

```
Create Table pedidos
(
id_pedido INT(4) NOT NULL,
id_cliente INT(4) NOT NULL,
id_articulo INT(4)NOT NULL,
fecha DATE,
```

cantidad INT(4),
total INT(4)
)

2. Restricción de Unicidad. La restricción **UNIQUE** obliga a que el contenido de una o más columnas no puedan repetir valores. la forma general para colocar esta restricción es la siguiente:

CREATE TABLE cliente(dni VARCHAR2(9) UNIQUE);

en el siguiente ejemplo, en la cración de la tabla pedidos, se incluye una restrición de unicidad. Create Table pedidos (id_pedido INT(4) NOT NULL UNIQUE, id_cliente INT(4) NOT NULL, id_articulo INT(4)NOT NULL, fecha DATE, cantidad INT(4), total INT(4))

Creación de una llave primaria

Una llave primaria, se define como un campo o conjunto(conbnacón de campos) que idetifican de forma unica un registro de una tabla. En el caso de que la llave primaria este conformada por un solo atributo, se crea de la siguiente forma:

Create Table pedidos
(
id_pedido INT(4) PRIMARY KEY,
id_cliente INT(4) NOT NULL,
id_articulo INT(4)NOT NULL,
fecha DATE,
cantidad INT(4),
total INT(4),
PRIMARY KEY(id_pedido)
)

En el caso en el cual se requiera que la llave primaria esta formada por varias columnas, la forma de crearla es la siguiente:

Create Table pedidos
(
id_pedido INT(4) NOT NULL AUTO_INCREMENT,
id_cliente INT(4) NOT NULL,
id_articulo INT(4)NOT NULL,
fecha DATE,
cantidad INT(4),
total INT(4),
CONSTRAINT PK_Pedido PRIMARY KEY (id_pedido,id_cliente,id_articulo)
)

Creación de llaves Foraneas

Una llave Foranea, se consiera como uno o mas campos de un tabla que hacen referencia al uno o varios campos que son llave primaria en otra tabla, indicando como estan relacionadas las tablas. Los datos en los campos de clave foranea y clave primaria deben coincidir en su tipo de dato, sin importar el nombre que tengan.

En el ejemplo de la tabla pedido, se debe hacer la referencia a la tabla clienes y a la tabla artículo, por lo cual se crean las llaver foraneas.

Create Table pedidos
(
id_pedido INT(4) NOT NULL AUTO_INCREMENT,

id_cliente INT(4) NOT NULL,
id_articulo INT(4)NOT NULL,
fecha DATE,cantidad INT(4),
total INT(4),
CONSTRAINT id_clente_fk REFERENCES clientes(dni),
CONSTRAINT id_articulo_fk REFERENCES articulo(id_articulo),
CONSTRAINT PK_Pedido PRIMARY KEY (id_pedido,id_cliente,id_articulo)
);

4.2.3 Sentencia de Modificación

La instrucción **ALTER TABLE** permite hacer cambios en la estructura de una tabla. por ejeplo añadir o borrar columnas, cambiar el tipo de columnas existentes o renombrar una columna.

la forma general de la instruccion es la siguiente:
ALTER TABLE nombreTabla ADD(nombreColumna TipoDatos
[Propiedades] [,columnaSiguiente tipoDatos
[propiedades]...)

con este tipo de instruccion, se pueden realiza las siguientes operaciones de modificacion:

Adicionar Columnas a la tabla

Si se tuviera la tabla Pedidos, y se necesitara aumentar un campo, que se llamara descripcion, se realizaria de la siguiente manera:

ALTER TABLE pedidos ADD (descripcion varchar);

Borrar una columna de una tabla

Para borrar una columna de una tabla, la forma general de la instruccion es la siguiente:

ALTER TABLE nombreTabla DROP(columna
[,columnaSiguiente,...]);

siguiendo el ejemplo anterior, si ahora se necesitara eliminar de la tabla pedidos el campo descripcion, se hace de la siguiente manera:
ALTER TABLE pedidos DROP (descripcion);

Cambiar el nombre de una columna

Para cambiar el nombre de una columna, la forma general de la instruccón es la siguiente:

ALTER TABLE nombreTabla CHANGUE columna tipodato,...

Siguiendo con elejemplo de la tabla pedido, si se requiere cambiar el nombre del campo cantidad por numero_elementos, se realizaria de la siguiente manera:

ALTER TABLE pedido CHANGE cantidad numero_elemetos INT (4);

4.3 STRUCTURED QUERY LANGUAGE (SQL) - DATA MANIPULATION LANGUAGE (DML)

El lenguaje de manipulación de datos (LMD) de SQL, agrupa los constructos necesarios tanto para la consulta como para la alteración de los datos almacenados en la base de datos. Ésta última incluye inserción, modificación y eliminación de tuplas en las tablas de las bases de datos.

4.3.1 Consultas

SQL provee algunos constructos que permiten imitar el comportamiento del Algebra Relacional (AR) para obtener información desde tablas, vistas o combinaciones de ambas. En esta sección se iniciará con la descripción de expresiones de consulta sencillas y se irá incrementando el nivel de complejidad a medida que se avanza en el tema. Además presenta las equivalencias entre las funciones del algebra relacional y los constructos SQL.

Para explicar el funcionamiento de las sentencias de consulta asumiremos la existencia de la siguiente tabla:

Tabla 4.43. Tabla para ejemplos

Productos		
Identificador	Nombre	Precio
300	Pera	700
301	Manzana	550
302	Arroz	1200
303	Zanahoria	1400

Consulta Básica

El formato más simple de la expresión de consulta en SQL es el siguiente:

SELECT * FROM {table | vista}

El formato más simple de la expresión de consulta en SQL es el siguiente:

SELECT * FROM {table | vista}

Aclaración: para interpretar la sintaxis de las expresiones en esta sección es necesario indicar:
- Los corchetes "[]" indican que lo que se encuentre entre ellos es opcional
- El carácter pipe "|" se puede leer como "o", es decir que puede ir uno u otro elemento
- Las llaves "{ }" indican inicio y fin de una expresion
- La expresión "[. . .]" indica que el elemento o conjunto de elementos anterior a ella se puede repetir indefinidamente

Se utiliza cuando se necesita obtener la información del total de columnas de la tabla o vista incluida en la cláusula FROM. La sentencia SQL para obtener todos los datos de todas las columnas de la tabla Productos será:

SELECT * FROM Productos

El equivalente en AR para el resultado del anterior código SQL incluiría solamente el nombre de la relación sin operador alguno, que debería presentar todas las columnas y tuplas que esta contiene.

Como se observó en algebra relacional es posible que de una relación se necesiten obtener sólo un conjunto determinado de columnas, para ello la sentencia de consulta SQL permite especificar cuáles de las columnas se desean desplegar en los resultados. Adicionando esta posibilidad la sintaxis de la consulta ahora es:

SELECT {* | {nombre_columna [as alias_columna]
[, nombre_columna [as alias_columna]] [. . .]} }
FROM {table | vista}

Donde cada "nombre_columna" corresponde al nombre de alguna de las columnas existentes en la tabla o vista de donde se extrae la información.

En caso que se deseen obtener solamente algunas de las columnas de la tabla Producto, se puede hacer uso de la sentencia indicando explícitamente cuales de las columnas se quieren desplegar, por ejemplo:

SELECT Nombre, Identificador FROM Productos

Retornará el siguiente resultado:

Tabla 4.44. Resultado consulta Simple

Nombre	Identificador
Pera	300
Manzana	301
Arroz	302
Zanahoria	303

El equivalente en álgebra relacional para la anterior consulta incluirá el operador "Proyección" de la siguiente forma:

$$\pi\ (nombre, identificador)(Productos)$$

Cabe resaltar que el orden de las columnas en el resultado depende del orden establecido en la sentencia de consulta y no en el orden de las columnas en las tablas.

Renombramiento de columnas

La sentencia de consulta permite establecerle un Alias a cada una de las columnas desplegadas mediante la palabra reservada AS, esto hará que en el resultado en lugar del nombre de la columna aparezca el alias establecido. Si el alias está compuesto por una única palabra se puede colocar sin comillas simples (‘), en otro caso (más de una palabra), éstas son obligatorias. En la siguiente sentencia por ejemplo se establecen alias a dos de las tres columnas de la tabla Productos:

SELECT Identificador AS Código, Precio,
Nombre AS ‘Nombre del producto’
FROM Productos

Retornará el siguiente resultado:

Tabla 4.45. Resultado de Consulta con Alias

Código	Precio	Nombre del producto
300	700	Pera
301	550	Manzana
302	1200	Arroz
303	1400	Zanahoria

El equivalente en álgebra relacional incluirá una proyección y un renombramiento, por ejemplo:

$$\rho\ Productos\ (código, precio, 'nombre\ del\ producto')(\ \pi(identificador, precio, nombre)(Productos))$$

Filtros a Consultas

Uno de los operadores mas utilizados del algebra relacional es la “Selección”, cuya funcionalidad es imitada por la cláusula WHERE dentro del constructo de consulta en SQL, si se adiciona dicha cláusula la sintaxis de la consulta en SQL queda de la siguiente forma:

SELECT {* | {nombre_columna [as alias_columna]
[, nombre_columna [as alias_columna]] [. . .]} }
FROM {tabla | vista}
[WHERE condición]

Donde la condición tiene el siguiente formato:

{ nombre_columna | Constante } {Comparador} {nombre_columna | Constante} [{Operador_lógico} {condicion}] [. . .]

Algunos de los comparadores más usados en SQL son los siguientes:
- Igualdad (=): retorna verdadero sólo cuando los dos operandos son exactamente iguales
- Diferencia (<>): retorna verdadero cuando los dos operandos son diferentes

- Mayor (>): retorna verdadero sólo cuando el primer operando es mayor que el segundo
- Menor(<): retorna verdadero sólo cuando el primer operando es menor que el segundo
- Mayor o igual (>=): retorna verdadero cuando el primer operando es mayor o igual que el segundo
- Menor o igual (<=):retorna verdadero cuando el primer operando es menor o igual que el segundo

En cuanto a los operadores lógicos SQL provee los siguientes:
- AND: retorna verdadero si las dos condiciones involucradas son verdaderas, retorna falso en cualquier otro caso
- OR: retorna verdadero si alguna de las dos condiciones involucradas son verdaderas, retorna falso si ambas son falsas
- NOT: niega el resultado de alguna condición

Las operaciones lógicas pueden anidarse un número ilimitado de veces mediante el uso de paréntesis, lo que permite crear lógicas booleanas muy complejas si se requieren.

Retomando el ejemplo con la tabla productos, podemos construir una sentencia que retorne solamente los productos con identificador mayor que 301, la consulta SQL sólo necesitará una sola condición como la siguiente:

```
SELECT *
FROM Productos
WHERE Identificador > 301
```

Este SQL retornará el siguiente resultado:

Tabla 4.46. Resultado de Consulta con Condicion Simple

Identificador	Nombre	Precio
302	Arroz	1200
303	Zanahoria	1400

El equivalente en AR es el siguiente:

$$\sigma\ identificador > 301\ (Productos)$$

Para ejemplificar la anidación de operaciones lógicas en una condición, el siguiente ejemplo de consulta SQL desplegará Identificador, Nombre renombrado como "nombre producto" y Precio de los productos cuyo Identificador sea menor o igual que 300 o que el nombre no sea "zanahoria" y tengan un precio mayor o igual a 1000:

```
SELECT Identificador as Codigo, Nombre as 'Nombre producto', Precio
FROM Productos
WHERE Identificador <= 300 OR (Nombre <> 'Zanahoria' AND Precio >= 1000)
```

La anterior sentencia retornará el siguiente resultado

Tabla 4.47. Resultado de Consulta con Condicion Compuesta

Identificador	Nombre producto	Precio
300	Pera	700
302	Arroz	1200

El equivalente de la anterior consulta SQL en AR es el siguiente:

$$\rho\ Productos(Identificador,'Nombre\ producto',precio)\left(\pi\ (Identificador, Nombre, Precio)\left(\sigma\ (Identificador \leq 300 \vee (nombre \neq' zanahoria' \wedge precio \geq 1000))(Productos)\right)\right)$$

Para los siguientes ejemplos se asumirá que la siguiente base de datos para un sistema que facilita el registro de compras en un almacén de abarrotes:

Figura 4.1: Base de datos Almacen de Abarrotes

Cientes	
Codigo	Nombre
701	Maritza Mera
702	Fabián Ruano
703	John Masso

Productos		
Identificador	Nombre	Precio
300	Pera	700
301	Manzana	550
302	Arroz	1200
303	Zanahoria	1400

Compras			
CódigoCliente	IdentificadorProducto	FechaHoraCompra	Cantidad
702	301	2013/10/29 1752	5
701	303	2013/11/01 10000	3
701	302	2013/11/02 1250	6
702	302	2013/11/02 1300	2

Cabe resaltar que en la base de datos las columnas CodigoCliente e IdentificadorProducto de la tabla Compras hacen referencia a la columna Código de la tabla Clientes y la columna Identificador de la tabla Productos respectivamente.

En las bases de datos relacionales es muy importante poder operar relaciones para obtener información en las consultas.

Para las operaciones básicas y extendidas del Algebra Relacional que permiten operaciones entre dos o más relaciones existen sus equivalentes en SQL, a continuación se harán explicitas esas equivalencias.

Producto Cartesiano

El soporte al producto cartesiano en SQL hace que la sintaxis del constructo de consulta se expanda para quedar de la siguiente forma:

```
SELECT {* | {[{tabla|vista}.]{nombre_columna} [as alias_columna]
[, [{tabla|vista}.]{nombre_columna} [as alias_columna]] [. . . ]}}
FROM {tabla | vista} [, {table | vista}] [. . . ]
[WHERE condición]
```

Se debe notar que en esta sintaxis a cada nombre_columna es opcional anteponerle el nombre de la tabla o vista de donde se extrae, esto porque puede suceder que en las tablas o vistas incluidas en el producto cartesiano exista más de una columna con el mismo nombre, lo que implica que se debe especificar de cual tabla o vista se extrae la información. En caso que el nombre de la columna sea único en todas las tablas y vistas involucradas en el producto no es necesario indicar de donde se extrae la información. Ocurre una situación similar en la condición de la sección WHERE, en donde será opcional incluir el nombre de la vista o tabla de donde se extrae la columna para evaluar la condición, la sintaxis de la condición ahora será:

{ [{tabla|vista}.]nombre_columna | Constante }
{Comparador} { [{tabla|vista}.]nombre_columna | Constante}
[{Operador_lógico} {condicion}] [. . .]

Se recomienda siempre especificar para cada columna en la consulta, el nombre o vista de donde se extraerá la información. Esto facilitará el mantenimiento de la consulta SQL y además evitará ambigüedad si las tablas o vistas de la base de datos cambian con el tiempo.

Por ejemplo, la siguiente consulta extrae todas las columnas resultado del producto cartesiano de las tablas Clientes y Productos:

SELECT * FROM Clientes, Productos

Su equivalente en Algebra relacional es:

$$Clientes \times Productos$$

En la siguiente consulta se extraen las columnas código de cliente e identificador de producto desde el producto cartesiano de las dos tablas:

SELECT Codigo, Identificador
FROM Clientes, Productos

El equivalente en Algebra relacional es:

$$\pi\ (código, identificador)(Clientes \times Productos)$$

En la siguiente consulta se adiciona la extracción de las columnas Nombre tanto de clientes como de productos, dado que ambas columnas se llaman igual se les debe anteponer el nombre de la tabla de origen y para mejorar la presentación se les colocará un alias a cada una de las columnas:

SELECT Codigo AS ‘Cliente Codigo’, Cliente.Nombre AS ‘Cliente Nombre’,
Identificador AS ‘Producto ID’, Producto.Nombre AS ‘Producto Nombre’
FROM Clientes, Productos

Cuyo resultado será:

Tabla 4.48. Resultado de consulta con Producto Cartesiano

Cliente Codigo	Cliente Nombre	Producto ID	Producto Nombre
701	Maritza Mera	300	Pera
701	Maritza Mera	301	Manzana
701	Maritza Mera	302	Arroz
701	Maritza Mera	303	Zanahoria
702	Fabián Ruano	300	Pera
702	Fabián Ruano	301	Manzana
702	Fabián Ruano	302	Arroz
702	Fabián Ruano	303	Zanahoria

Su equivalente en álgebra relacional es:

$$\rho\,('Cliente\ Codigo','Cliente\ Nombre','ProductoID','ProductoNombre')\,(\pi\ Codigo,$$
$$Clientes.Nombre, Identificador, Productos.Nombre\,(Clientes\ \times Productos))$$

Alias en Tablas o Vistas

Para las operaciones que involucran una sola tabla de ambos lados del operador y para mejorar la legibilidad de la consulta es posible establecerle un Alias a la tabla o vista, el cual deberá ser utilizado para nombrar a las columnas tanto en la sección SELECT como en la condición en el WHERE. La sintaxis con este nuevo elemento queda así:

SELECT {* | {[{tabla|vista|alias}.]{nombre_columna} [as alias_columna]
[,[{tabla|vista|alias}.]{nombre_columna} [as alias_columna]] [. . .] }}
FROM {tabla | vista} [alias] [, {tabla | vista} [alias]] [. . .]
[WHERE condición]

Donde la sintaxis de la condición es:
{ [{tabla|vista|alias}.]nombre_columna | Constante }
{Comparador} { [{tabla|vista|alias}.]nombre_columna | Constante}
[{Operador_lógico} {condicion}] [. . .]

La siguiente consulta presenta los nombres de clientes y productos extraídos de la operación producto cartesiano entre la selección de clientes con Código menor o igual que 701 y la selección de productos con identificador mayor que 301. Se utilizan Alias para renombrar las tablas los cuales deben ser antepuestos tanto en la sección SELECT como en el WHERE.

SELECT Cli.Nombre AS ‘Nombre Cliente’, Pro.Nombre AS ‘Nombre Producto’
FROM Productos Pro, Clientes Cli
WHERE Pro.Identificador > 301 AND Cli.Codigo >= 701

La anterior consulta dará como resultado:

Tabla 4.49. Resultado de Consulta Condicionada

Nombre Cliente	Nombre Producto
Maritza Mera	Arroz
Maritza Mera	Zanahoria

Su equivalente en Algebra Relacional:

$$\rho\ (\text{'Nombre Cliente'}, \text{'Nombr Producto'})\ (\pi\ cli.Nombre, pro.Nombre\ (\sigma\ Codigo \geq 701(\ \rho\ cli(Clientes)) \times \sigma\ \text{Identificador} > 301\ (\rho\ \text{pro}(\ Productos)\ )\ )\ )$$

Unión

En SQL el operador Unión del algebra relacional funciona a nivel de consultas, la sintaxis del operador es:

Sentencia_Consulta UNION Sentencia_Consulta

Donde cada Sentencia_Consulta es un constructo que incluye SELECT, FROM y WHERE. Son condiciones para poder ejecutar la unión entre dos consultas:
- Que el número de columnas resultado de ambas consultas coincida
- El tipo de la i-ésima columna de la primera consulta coincide con el tipo de la i-ésima columna de la segunda consulta involucrada en la Unión, para cada i desde 1 hasta el número de columnas de las consultas

El siguiente ejemplo SQL une el resultado de las siguientes dos consultas:
Nombres y precios de los productos con costo menor que 700
Nombres y precios de los productos con identificador mayor que 301

```
SELECT Nombre, Precio
FROM Productos
WHERE Precio < 600
UNION
SELECT Nombre, Precio
FROM Productos
WHERE Identificador > 301
```

En Algebra Relacional la consulta se expresaría de la siguiente forma:

$$\pi\ (Nombre, Precio)\ (\ \sigma\ Precio < 600\ (Productos))\ \cup\ \pi\ (Nombre, Precio)(\sigma\ Identificador > 301\ (Productos)\ )$$

El resultado de la consulta es:

Figura 4.2: Ejemplo de funcionamiento de UNION en SQL

Nombre	Precio
Pera	550

U

Nombre	Precio
Arroz	1200
Zanahoria	1400

=

Nombre	Precio
Pera	550
Arroz	1200
Zanahoria	1400

Diferencia

Terminando con el mapeo de las operaciones básicas del algebra relacional tenemos la Diferencia, en SQL representada por el operador MINUS cuya sintaxis es:

Sentencia_Consulta MINUS Sentencia_Consulta

Para que dos sentencias de consulta puedan operarse mediante la diferencia es necesario cumplir las mismas condiciones requeridas por el operador UNION.

El siguiente ejemplo SQL presenta los Identificadores de los productos que no están en la tabla Compras

```
SELECT Identificador
FROM PRODUTO
MINUS
SELECT IdentificadorProducto AS Identificador
FROM Compras
```

En algebra relacional la sentencia correspondiente es:

$$\pi\, Identificador(Productos) - \rho\, (Identificador)(\pi\, (IdenficadorProducto)(Compras))$$

El resultado de la consulta es:

Figura 4.3: Ejemplo de funcionamiento de MINUS en SQL

Idenficador
300
301
302
303

−

Identificador
301
302
303

=

Identificador
300

Intersección

El constructo de consulta provisto por SQL también soporta Intersección entre consultas, la sintaxis es la siguiente:
Sentencia_Consulta INTERSECT Sentencia_Consulta
Al igual que los operadores de unión y diferencia las sentencias involucradas deben tener igual número de columnas y compatibilidad uno a uno en tipos de datos.
El siguiente ejemplo SQL presenta los identificadores de productos registrados tanto en la tabla Compras como en la tabla Productos:

```
SELECT Identificador
FROM PRODUTO
INTERSECT
SELECT IdentificadorProducto AS Identificador
FROM Compras
```

En algebra relacional la sentencia correspondiente es:

$$\pi\, Identificador(Productos) \cap \rho\, (Identificador)(\pi\, (IdenficadorProducto)(Compras))$$

El resultado de la consulta es:

Figura 4.4: Ejemplo funcionamiento INTERSECT en SQL

Idenficador
300
301
302
303

∩

Identificador
301
302
303

=

Identificador
301
302
303

Unión Natural

Una de las funciones más útiles en SQL es la Unión en sus diferentes versiones, Natural, Completa, "Por Derecha", "Por Izquierda" e "Interna". La Unión Natural combina la proyección, selección y producto cartesiano en una sola operación, adiciona una condición implícita a la operación igualando la llave primaria a la llave foránea (debe existir una restricción del tipo llave foránea –ForeignKey- entre las tablas operadas), y en la proyección elimina la columna, el soporte a esta función hace que la sintaxis de la consulta SQL quede de la siguiente forma:

```
SELECT {* |
{[{tabla|vista|alias}.]{nombre_columna} [as nombre_presentar1]
[,[{tabla|vista|alias}.]{nombre_columna} [as nombre_presentar2]] [. . . ] }}
FROM {tabla | vista} [alias] [, {tabla | vista} [alias]] [. . . ]
[NATURAL JOIN {tablaN | vista} [alias]][. . . ]
[WHERE condición]
```

La siguiente consulta SQL presenta el nombre de los productos, fecha de compra, identificadores de productos y cantidad comprada por cada uno de los clientes.

```
SELECT Producto.Nombre, Compras.IdentificadorProducto,
Compras.FechaHoraCompra, Compras.Cantidad
FROM Producto NATURAL JOIN Compras
```

El equivalente en álgebra relacional será:

$$\pi\,(Productos.Nombre, Compras.IdentificadorProducto, Compras.FechaHoraCompra, Compras.Cantidad)(Productos \bowtie Compras)$$

Y el resultado de la consulta es:

Tabla 4.50. Resultado de consulta Natural Join o Union Natural

Nombre	IdentificadorProducto	FechaHoraCompra	Cantidad
Manzana	301	2013/10/29 1752	5
Zanahoria	303	2013/11/01 10000	3
Arroz	302	2013/11/02 1250	6
Arroz	302	2013/11/02 1300	2

Unión por derecha y por izquierda

Las Uniones por derecha o por izquierda hacen que la sintaxis de la consulta en SQL se ajuste para adicionar la condición de unión de las tablas o vistas.

SELECT {* |
{[{tabla|vista|alias}.]{nombre_columna} [as nombre_presentar1]
[,[{tabla|vista|alias}.]{nombre_columna} [as nombre_presentar2]] [. . .] }}
FROM {tabla | vista} [alias] [, {tabla | vista} [alias]] [. . .]
[NATURAL JOIN {tabla | vista} [alias]] [. . .]
[{LEFT | RIGHT} JOIN {tabla | vista} [alias] ON condicion] [. . .]
[WHERE condición]

En el siguiente ejemplo SQL se hace una Unión por la Izquierda entre Productos y Compras, lo que garantiza que se desplieguen todos los productos independientemente que estén relacionados o no en la tabla Compras. Se extrae nombre del producto, identificador, fecha y hora de compra y cantidad comprada.

SELECT Producto.Nombre AS NombreP, Compras.IdentificadorProducto,
Compras.FechaHoraCompra, Compras.Cantidad
FROM Producto LEFT JOIN Compras
On Producto.Identificador = Compras.IdentificadorProducto

El resultado de la consulta es:

Tabla 4.51. Resultado de Consulta con LEFT JOIN o Union por la Izquierda

NombreP	IdentificadorProducto	FechaHoraCompra	Cantidad
Manzana	301	2013/10/29 1752	5
Zanahoria	303	2013/11/01 10000	3
Arroz	302	2013/11/02 1250	6
Arroz	302	2013/11/02 1300	2
Pera	NULL	NULL	NULL

En el siguiente ejemplo se realiza una Unión por la Derecha entre Compras y Clientes lo que hace que se presenten todos los clientes independientemente de si están o no relacionados con alguna tupla de la tabla Compras:

SELECT Compras.CodigoCliente, Compras.FechaHoraCompra,
Compras.Cantidad, Clientes.Nombre AS NombreC
FROM Compras RIGTH JOIN Clientes
On Clientes.Codigo = Compras.CodigoCliente
El resultado de la consulta es:

Tabla 4.52. Resultado de Consulta con RIGTH JOIN o Union por la Derecha

IdentificadorProducto	FechaHoraCompra	Cantidad	NombreC
301	2013/10/29 1752	5	Fabian Ruano
303	2013/11/01 10000	3	Maritza Mera
302	2013/11/02 1250	6	Maritza Mera
302	2013/11/02 1300	2	Fabian Ruano
NULL	NULL	NULL	John Masso

Igual que en álgebra relacional, en SQL, el resultado de las operaciones con LEFT JOIN y RIGTH JOIN depende de la posición de los operandos. Ambos operadores presentan el total de registros de la tabla o vista a la izquierda o derecha respectivamente sin importar si la existe un registro relacionado en la otra tabla o vista operada.

Unión Completa

La Unión Completa expande aún más la sintaxis de la consulta quedando de la siguiente forma:

```
SELECT {* |
{[{tabla|vista|alias}.]{nombre_columna} [as nombre_presentar1]
[,[{tabla|vista|alias}.]{nombre_columna} [as nombre_presentar2]] [. . . ] }}
FROM {tabla | vista} [alias] [, {tabla | vista} [alias]] [. . . ]
[NATURAL JOIN {tabla | vista} [alias]] [. . . ]
[{LEFT | RIGHT | FULL} JOIN {tabla | vista} [alias]
ON condición] [. . . ]
[WHERE condición]
```

En el siguiente ejemplo SQL se hace uso de la unión completa para extraer toda la información de la base de datos de ejemplo uniendo las tres tablas y garantizando que se presente toda la información de las tablas productos y clientes aunque no estén relacionadas en la tabla compras. Cabe anotar que el siguiente ejemplo de unión completa utiliza la notación de ORACLE, no se garantiza su fucionamiento en otros motores de bases de datos.

```
SELECT Clientes.Nombre AS Cliente,
Compras.FechaHoraCompra, Compras.Cantidad,
Productos.Nombre AS Producto
FROM Clientes
FULL JOIN Compras ON Cientes.Codigo = Compras.CodigoCliente
FULL JOIN Productos ON Productos.Identificador = Compras.IdentificadorProducto
```

Tabla 4.53. Resultado de consulta con FULL JOIN o Union Completa

Cliente	FechaHoraCompra	Cantidad	Producto
Fabian Ruano	2013/10/29 1752	5	Manzana
Maritza Mera	2013/11/01 10000	3	Zanahoria
Maritza Mera	2013/11/02 1250	6	Arroz
Fabian Ruano	2013/11/02 1300	2	Arroz
John Masso	NULL	NULL	NULL
NULL	NULL	NULL	Pera

Unión Interna

La Unión Interna garantiza que en el resultado de la consulta solamente se presenten los elementos que cumplen con la condición de unión. Al incluir este nuevo elemento en la sintaxis de la consulta SQL quedará:

```
SELECT {* |
{[{tabla|vista|alias}.]{nombre_columna} [as nombre_presentar1]
[,[{tabla|vista|alias}.]{nombre_columna} [as nombre_presentar2]] [. . . ] }}
FROM {tabla | vista} [alias] [, {tabla | vista} [alias]] [. . . ]
[NATURAL JOIN {tabla | vista} [alias]] [. . . ]
[{LEFT | RIGHT | FULL | INNER} JOIN {tabla | vista} [alias]
ON condición] [. . . ]
[WHERE condición]
```

En el siguiente ejemplo se presenta una unión interna entre las tablas Clientes, Compras y Productos condicionadas por la igualdad entre identificadores de las tablas:

SELECT Clientes.Nombre AS Cliente,
Compras.FechaHoraCompra, Compras.Cantidad,
Productos.Nombre AS Producto
FROM Clientes
INNER JOIN Compras ON Cientes.Codigo = Compras.CodigoCliente
INNER JOIN Productos ON Productos.Identificador = Compras.IdentificadorProducto

El siguiente es el resultado de la consulta:

Tabla 4.54. Resultado de consulta con INNER JOIN o Union Interna

Cliente	FechaHoraCompra	Cantidad	Producto
Fabian Ruano	2013/10/29 1752	5	Manzana
Maritza Mera	2013/11/01 10000	3	Zanahoria
Maritza Mera	2013/11/02 1250	6	Arroz
Fabian Ruano	2013/11/02 1300	2	Arroz

En todas las uniones que necesitan una condición (derecha, izquierda, completa e interna), la condición puede incluir comparaciones entre dos columnas, usar operadores lógicos, anidar condiciones e incluir todas las columnas de las tablas previamente mencionadas en la cláusula FROM de la consulta.

Funciones de grupo

La Agrupación permite formar grupos entre los datos obtenidos en la consulta para luego realizar cálculos matemáticos con algunas columnas por cada uno de los grupos formados, entre los posibles cálculos a realizar están: Suma, Promedio, Conteo, Máximo y Mínimo. En SQL la función es soportada por la cláusula GROUP BY, cuando se utiliza esta cláusula la sintaxis de la consulta es:

SELECT
{ [{tabla|vista|alias}.]{nombre_columna} [as alias_columna] |
{AVG | SUM | COUNT | MAX | MIN} ([{tabla|vista|alias}.]{nombre_columna})
[as alias_columna] }
[,[{tabla|vista|alias}.]{nombre_columna} [as alias_columna] |
{AVG | SUM | COUNT | MAX | MIN} ([{tabla|vista|alias}.]{nombre_columna})
[as alias_columna]] [. . .]
FROM {tabla | vista} [alias] [, {tabla | vista} [alias]] [. . .]
[NATURAL JOIN {tablaN | vista} [alias]] [. . .]
[{LEFT | RIGHT | FULL | INNER} JOIN {tabla | vista} [alias]
ON condición] [. . .]
[WHERE condición]
GROUP BY [{tabla|vista|alias}.]{nombre_columna}
[, [{tabla|vista|alias}.]{nombre_columna}] [. . .]

En la anterior sintaxis debe notarse que las funciones matemáticas disponibles para operar las columnas son:

- AVG, que obtiene el promedio matemático de a columna para las tuplas que pertenecen a cada uno de los grupos

- SUM, que realiza una suma de los valores de la columna para las tuplas de cada uno de los grupos
- COUNT, realiza un conteo de las tuplas de cada uno de los grupos
- MAX, retorna el máximo valor de la columna para cada uno de los grupos
- MIN, retorna el mínimo valor de la columna para cada uno de los grupos

En caso que alguno de los valores de la columna operada por las anteriores funciones sea NULL, este no se tomará en cuenta para la operación.
Cuando se utiliza la cláusula GROUP BY en una consulta SQL en la sección SELECT sólo pueden incluirse dos tipos de elementos: las columnas que generan los grupos (que corresponden a las columnas indicadas en la sección GROUP BY) y las funciones matemáticas previamente descritas.

El siguiente ejemplo obtiene el total de productos comprados por cada cliente sumando las cantidades registradas en la tabla compras por cada uno de ellos:

SELECT Clientes.Nombre, SUM(Compras.Cantidad) AS Total
FROM Compras NATURAL JOIN Clientes
GROUP BY Clientes.Nombre

La anterior consulta opera de la siguiente forma:

1. Realiza las operaciones de la cláusula FROM y aplica las restricciones de la cláusula WHERE (Si ésta se encuentra definida), luego de este primer paso se tendría algo como lo siguiente:

Tabla 4.55. Ejemplo funcionamiento de Group by, paso 1

IdentificadorProducto	FechaHoraCompra	Cantidad	Código	Nombre
702	2013/10/29 1752	5	702	Fabian Ruano
701	2013/11/01 10000	3	701	Maritza Mera
701	2013/11/02 1250	6	701	Maritza Mera
702	2013/11/02 1300	2	702	Fabian Ruano

2. Se identifican los grupos teniendo en cuenta las columnas seleccionadas en la cláusula GROUP BY. Para el ejemplo, dado que en la columna Clientes.Nombre existen dos valores, se identifican dos grupos:

Tabla 4.56. Ejemplo funcionamiento de Group by, paso 2

IdentificadorProducto	FechaHoraCompra	Cantidad	Código	Nombre
702	2013/10/29 1752	5	702	Fabian Ruano
701	2013/11/01 10000	3	701	Maritza Mera
701	2013/11/02 1250	6	701	Maritza Mera
702	2013/11/02 1300	2	702	Fabian Ruano

3. Se realizan las operaciones por cada grupo de la consulta, para el grupo de “Fabian Ruano” la suma de los valores de la columna cantidad es igual a siete (7) y para el grupo de “Maritza Mera” es igual a nueve (9).

4. Finalmente se retornan los valores de cada columna para cada grupo y los de las columnas calculadas:

Tabla 4.57. Ejemplo funcionamiento de Group by, paso 4

Nombre	Total
Fabian Ruano	7
Maritza Mera	9

El siguiente ejemplo SQL presenta un listado con cantidad de compras realizadas, promedio de productos por compra y máximo de productos por compra y mínimo de productos por compra para cada cliente. La consulta incluye a todos los clientes incluso si no han realizado compras, en tal caso presenta las operaciones como cero (0).

```
SELECT Clientes.Nombre, COUNT(*) AS Cuenta, AVG(Compras.Cantidad) AS Promedio,
MAX(Compras.Cantidad) AS Maximo,
MIN(Compras.Cantidad) AS Minimo
FROM Compras
RIGHT JOIN Clientes ON Compras.CodigoCliente = Clientes.Codigo
GROUP BY Clientes.Nombre
```

El resultado de la consulta es el siguiente:

Tabla 4.58. Ejemplo consulta con operadores

Nombre	Cuenta	Promedio	Maximo	Minimo
Maritza Mera	2	4.5	6	3
Fabian Ruano	2	3.5	5	2
John Masso	0	0	0	0

Orden en la Consulta

Hasta ahora el Orden de presentación de los resultados en la ejecución de una consulta SQL depende del orden que se hayan almacenado en la base de datos o cualquier orden que el intérprete SQL que se esté utilizando le desee dar. La sentenciaORDER BY permite ordenar descendente o ascendentemente teniendo en cuenta una o más columnas de las tablas o vistas involucradas en la consulta. La sintaxis con este nuevo elemento queda así:

```
SELECT {*|
{ [{tabla|vista|alias}.]{nombre_columna} [as alias_columna] |
{AVG | SUM | COUNT | MAX | MIN} ([{tabla|vista|alias}.]{nombre_columna})
[as alias_columna] }
[,[{tabla|vista|alias}.]{nombre_columna} [as alias_columna] |
{AVG | SUM | COUNT | MAX | MIN} ([{tabla|vista|alias}.]{nombre_columna})
[as alias_columna] ] [...] }
FROM {tabla | vista} [alias] [, {tabla | vista} [alias]] [...]
[NATURAL JOIN {tabla | vista} [alias]] [...]
[{LEFT | RIGHT | FULL | INNER} JOIN {tabla | vista} [alias]
ON condición] [...]
[WHERE condición]
[GROUP BY [{tabla|vista|alias}.]{nombre_columna}
[, [{tabla|vista|alias}.]{nombre_columna}] [...] ]
[ORDER BY [{tabla|vista|alias}.]{nombre_columna} [DESC]
```

[, [{tabla|vista|alias}.]{nombre_columna} [DESC]] [. . .]]

Para ejemplificar el funcionamiento del ORDER BY se propone la siguiente consulta:

SELECT Clientes.Nombre AS Cliente, Compras.Cantidad,
Productos.Nombre AS Producto
FROM Clientes
INNER JOIN Compras ON Cientes.Codigo = Compras.CodigoCliente
INNER JOIN Productos ON Productos.Identificador = Compras.IdentificadorProducto

Que reporta el siguiente resultado:

Tabla 4.59. Ejemplo de consulta con ORDER BY

Cliente	FechaHoraCompra	Cantidad	Producto
Fabian Ruano	2013/10/29 1752	5	Manzana
Maritza Mera	2013/11/01 10000	3	Zanahoria
Maritza Mera	2013/11/02 1250	6	Arroz
Fabian Ruano	2013/11/02 1300	2	Arroz

Para ordenar los registros de la consulta por nombre de cliente y cantidad de productos comprados se deberá modificar para que quede:

SELECT Clientes.Nombre AS Cliente, Compras.Cantidad,
Productos.Nombre AS Producto
FROM Clientes
INNER JOIN Compras ON Cientes.Codigo = Compras.CodigoCliente
INNER JOIN Productos
ON Productos.Identificador = Compras.IdentificadorProducto
ORDER BY Clientes.Nombre, Compras.Cantidad DESC

El resultado será:

Tabla 4.60. Ejemplo de consulta con ordenamiento descendente

Cliente	FechaHoraCompra	Cantidad	Producto
Fabian Ruano	2013/10/29 1752	5	Manzana
Fabian Ruano	2013/11/02 1300	2	Arroz
Maritza Mera	2013/11/02 1250	6	Arroz
Maritza Mera	2013/11/01 10000	3	Zanahoria

Debe notarse que luego de ordenar los registros por el nombre del cliente de forma ascendente se ordenaron por cantidad comprada de forma descendente. Si no se especifica el ordenamiento descendente con el constructo DESC por defecto se realiza ordenamiento ascendente.

4.3.2 Sub-Consultas

Como seguramente ya se habrá notado en la sección anterior, el resultado de una consulta siempre será una tabla, pues bien, dichos resultados pueden ser incluidos en la sección FROM de consultas de orden superior para ser usadas como fuentes de datos. En palabras simples: se pueden realizar consultas sobre resultados de otras consultas.

En esta sección se llamará consulta de orden superior a toda consulta que incluya una o más

sub-consultas. La sintaxis de las consultas de orden superior queda así:

```
SELECT {*|
{ [{tabla|vista|alias}.]{nombre_columna} [as alias_columna] |
{AVG | SUM | COUNT | MAX | MIN} ([{tabla|vista|alias}.]{nombre_columna}) [as alias_columna]
}
[,[{tabla|vista|alias}.]{nombre_columna} [as alias_columna] |
{AVG | SUM | COUNT | MAX | MIN} ([{tabla|vista|alias}.]{nombre_columna})
[as alias_columna] ] [. . . ] }
FROM {tabla | vista | (sub-consulta)} [alias] [, {tabla | vista} [alias]] [. . . ]
[NATURAL JOIN {tabla | vista | (sub-consulta)} [alias]] [. . . ]
[{LEFT | RIGHT | FULL | INNER} JOIN {tabla | vista | (sub-consulta)}
[alias] ON condición] [. . . ]
[WHERE condición]
[GROUP BY [{tabla|vista|alias}.]{nombre_columna}
[, [{tabla|vista|alias}.]{nombre_columna}] [. . . ] ]
[ORDER BY [{tabla|vista|alias}.]{nombre_columna} [DESC]
[, [{tabla|vista|alias}.]{nombre_columna} [DESC]] [. . . ] ]
```

Se deben aclarar varios puntos respecto a la utilización de sub-consultas:

- Cada sub-consulta puede incluir todos los elementos de una consulta SQL de orden superior Excepto por la cláusulaORDER BY.
- El orden de presentación de los resultados de las sub-consultas no afecta en ninguna medida las operaciones de la cláusula FROM de la consulta de orden superior.
- El orden de presentación de los resultados dependerá únicamente de la cláusula ORDER BY de la consulta de orden superior.
- Siempre que se utilice una sub-consulta es Obligatorio el uso de un Alias que permita acceder a las columnas resultado de dicha sub-consulta

El siguiente ejemplo hace uso de sub consultas para obtener el listado total de compras realizado por los clientes con código menor que 702.

```
SELECT Alfa.NC AS 'Nombre Cliente', Compras.IdentificadorProducto,
Compras.FechaHoraCompra, Compras.Cantidad
FROM (
SELECT Codigo AS IdCliente, Nombre AS NC
FROM Clientes
WHERE Codigo < 702
) AS Alfa
INNER JOIN Compras ON Alfa.IdCliente = Compras.CodigoCliente
```

La sub-consulta renombrada como Alfa obtiene el listado de clientes con que cumplan la condición, dentro de ella además se renombran las columnas:

Tabla 4.61. Resultado sub-consulta

Alfa	
IdCliente	NC
701	Maritza Mera

Alfa se une con la tabla Compras y el resultado de la consulta es:

Tabla 4.62. Resultado de consulta con sub-consulta

Nombre Cliente	IdentificadorProducto	FechaHoraCompra	Cantidad
Maritza Mera	303	2013/11/01 10000	3
Maritza Mera	302	2013/11/02 1250	6

Existen operadores especiales que permiten hacer comparaciones entre una columna y un listado de valores, los operadores usados para ello son:
- ALL: retorna verdadero cuando se cumple que la comparación entre el valor de la columna y cada uno de los elementos de la lista es verdadera.
- ANY: retorna verdadero cuando se cumple que la comparación entre el valor de la columna y al menos uno de los elementos de la lista es verdadera.
- IN: retorna verdadero cuando el valor de la columna específica está en el listado.

El siguiente es un ejemplo de uso del operador IN, obtienen de la lista de empleados solamente aquellas tuplas cuyo código este en el listado {701, 703}.

```
SELECT *
FROM Clientes
WHERE Clientes.Codigo IN (701, 703)
```

El resultado es:

Tabla 4.63. Resultado de consulta con operador IN

Código	Nombre
701	Maritza Mera
703	John Masso

A continuación se utiliza el operador ANY para obtener la información de los productos que tengan precio mayor que alguno de los precios del siguiente listado: {750, 1200}

```
SELECT *
FROM Productos
WHERE Precio > ANY (750, 1200)
```

El resultado es:

Tabla 4.64. Resultado de consulta con operador ANY

Identificador	Nombre	Precio
302	Arroz	1200
303	Zanahoria	1400

Para obtener los productos que tengan precio mayor que todos los elementos del listado: {750,1200} bastará con modificar la consulta reemplazando ANY por ALL.

```
SELECT *
FROM Productos
WHERE Precio > ALL (750, 1200)
```

El resultado es:

Tabla 4.65. Resultado de consulta con operador ALL

Identificador	Nombre	Precio
303	Zanahoria	1400

Los listados para comparación pueden ser provistos por resultados de sub-consultas que retornen una sola columna. Por ejemplo, el siguiente SQL retorna el listado de precios para los productos que haya comprado el cliente con código 701 haciendo uso de sub-consultas para la condición:

```
SELECT *
FROM PRODUCTO
WHERE Identificador IN (
SELECT IdenficadorProducto
FROM Compras
WHERE CodigoCliente = 701)
```

La sub-consulta retorna el siguiente resultado:

Tabla 4.66. Resultado de consulta con operador IN y Sub-consulta

IdentificadorProducto
303
302

4.3.3 Inserción

La cláusula INSERT del LMD permite realizar la inserción de nuevos registros en las tablas, previamente creadas, de la base de datos. El formato general de la sentencia es:

INSERT INTO <nombre de tabla> [(campo1 [, campo2] [. . .])]
VALUES (valor1 [,valor2] [. . .])

Por ejemplo, dada la siguiente tabla de una base de datos

Tabla 4.67. Definición de tabla para ejemplos de inserción

Personal		
Identificador: INTEGER	NOT NULL	PrimaryKey
Nombres: VARCHAR(50)		
Apellidos: VARCHAR(50)	NOT NULL	

Se podría insertar una tupla a dicha tabla con la siguiente sentencia:
INSERT INTO Personal (Identificador, Apellidos) VALUES (501, 'Gómez')

Luego de la ejecución de dicha sentencia nuestra tabla Personal quedará con los siguientes valores:

Tabla 4.68. Resultado de inserción

Personal		
Identificador	Nombres	Apellidos
501	NULL	Gómez

Nótese que el campo Nombres se obvió de la instrucción, esto se puede hacer porque dicho campo permite valores NULL (no es obligatorio), en ese orden de ideas la siguiente sentencia es INCORRECTA:
INSERT INTO Personal (Apellidos, Nombres) VALUES ('Gómez', 'María')

Al no suministrar un valor para el campo Identificación, la sentencia no satisface la restricción de obligatoriedad de dicho campo. Entonces para construir una sentencia de inserción de una tupla a una tabla es necesario tener en cuenta cuales de los atributos son obligatorios.
El constructo INSERT permite realizar la inserción sin especificar los campos que se van a tener en cuenta en la consulta.
INSERT INTO Personal VALUES (502, 'López', 'Pedro')

En este caso el intérprete SQL asume que los valores deben insertarse teniendo en cuenta el orden en que fueron suministrados en la sentencia y el orden en que fueron creados los campos a la hora de definir la tabla, por tanto, luego de ejecutar la anterior consulta la tabla tendría la siguiente información.

Tabla 4.69. Resultado de inserción sin especificación de campos

Personal		
Identificador	Nombres	Apellidos
501	NULL	Gómez
502	López	Pedro

Esto es: el primer valor enviado se intentará asignar a la primera columna de la tabla, el segundo valor a la segunda columna y así sucesivamente. Entonces, si se decide usar esta sintaxis para la sentencia INSERT se debe tener en cuenta el orden de la creación de los atributos de la tabla.

Qué pasaría con las sentencias INSERT que no incluyen especificación de campos si. . . .

1. Luego de creada la tabla decido eliminar una de las columnas?
2. Luego de la operación anterior inserto una columna nueva?

En la cláusula INSERT también se deben tener en cuenta los tipos de datos de las columnas “destino”. Usualmente los tipos de datos numéricos (integer, float, decimal, etc.), se insertan directamente en la sentencia y en caso del carácter de separación decimal se utiliza el punto (.) debido a que la coma (,) hace parte de la sintaxis de la cláusula.

Por ejemplo, teniendo la tabla:

Tabla 4.70. Tabla ejemplo para acciones de inserción

Factura
NumeroFactura: INTEGER NOT NULL PrimaryKey
TotalFactura: DECIMAL (18,2)
IVA: FLOAT
Vendedor: VARCHAR(100)
FechaCompra: DATETIME

La siguiente sentencia será válida:

INSERT INTO Factura (NumeroFactura, TotalFactura, IVA)
VALUES (401, 54500, 0.16)

De otro lado los valores para los tipos no-numéricos (char, varchar, datetime, etc.) requieren estar delimitados por comillas simples (‘) cuando se incluyen en una sentencia INSERT.

Para la misma tabla Factura la siguiente sentencia será válida:

INSERT INTO Factura
VALUES (402, 34290, 0.17, ‘Ruben Rodriguez’, ‘2013/05/24 15:00’)

4.3.4 Borrado

Para el borrado de una o más tuplas almacenadas en las tablas de la base de datos en SQL existe el constructo DELETE con el siguiente formato general:
DELETE FROM <nombre tabla> [WHERE <condición>]

La condición de la sentencia tiene la misma sintaxis de las condiciones en el constructo para consultas, las columnas a comparar serán las columnas de la tabla sobre la que se pretender realizar la operación de borrado. Si la condición no se incluye se asume que se van a borrar todas las tuplas de la tabla.
Para los siguientes ejemplos se utilizara la tabla Factura con los siguientes datos y columnas:

Tabla 4.71. Tabla ejemplo para acciones de borrado

FACTURA				
Numero	Total	IVA	Vendedor	Fecha Compra
401	55000	8800	Sandra Roa	28/05/2013
402	45000	7200	Maria Vidal	31/05/2013
403	60000	9600	Maria Vidal	20/08/2013
404	35000	5600	Sandra Roa	20/09/2013

La sentencia:
DELETE FROM Factura

Eliminaría todas las tuplas existentes en la tabla.

Haciendo uso de condicionales simples, con la sentencia:

DELETE FROM Factura WHERE Total > 50000

Se borrarían dos de los registros de la tabla y la nueva tabla sería:

Tabla 4.72. Resultado de borrado de registros con condición

FACTURA				
Numero	Total	IVA	Vendedor	Fecha Compra
402	45000	7200	Maria Vidal	31/05/2013
404	35000	5600	Sandra Roa	20/09/2013

Haciendo uso de condicionales con listas es posible eliminar elementos con un listado constante como el siguiente: {7200, 9600}

DELETE FROM Factura WHERE IVA IN {7200, 9600}

Dejando como resultado una tabla con los siguientes datos:

Tabla 4.73. Resultado de borrado de registros con operador IN

FACTURA				
Numero	Total	IVA	Vendedor	Fecha Compra
401	55000	8800	Sandra Roa	28/05/2013
404	35000	5600	Sandra Roa	20/09/2013

Es posible utilizar sub-consultas para seleccionar los elementos a borrar, por ejemplo en el siguiente SQL se eliminan todos los registros cuyo IVA sea mayor al IVA de todos las compras vendidos por "Sandra Roa"

DELETE FROM Factura
WHERE IVA > ALL (
SELECT IVA
FROM Factura
WHERE Vendedor = 'Sandra Roa')

Luego de la ejecución de la sentencia la tabla Factura quedará así:

Tabla 4.74. Resultado de borrado de registros con operador ALL

FACTURA				
Numero	Total	IVA	Vendedor	Fecha Compra
401	55000	8800	Sandra Roa	28/05/2013
402	45000	7200	Maria Vidal	31/05/2013
404	35000	5600	Sandra Roa	20/09/2013

4.3.5 Actualización

SQL provee constructos para la actualización de datos existentes en una tabla de la base de datos, la sintaxis de la sentencia es:

UPDATE {Tabla}
SET {NombreColumna} = {NuevoValor} [,{NombreTabla} = {NuevoValor}] [. . .]
[WHERE {condicion}]

La condición de la sentencia tiene la misma sintaxis de las condiciones en el constructo para consultas, las columnas a comparar serán las columnas de la tabla sobre la que se pretender realizar la actualización.

La sentencia establece los nuevos valores a las columnas de todas las tuplas que cumplan con la condición, si la condición se omite se asume que se van a actualizar todas las tuplas de la tabla.

El NuevoValor puede ser una constante o el resultado de operaciones entre diferentes tipos de datos (operaciones matemáticas, operaciones de cadenas, etc).

Para los siguientes ejemplos se utilizara la tabla Factura con los siguientes datos y columnas:

Tabla 4.75. Tabla ejemplo para acciones de actualización

FACTURA				
Numero	Total	IVA	Vendedor	FechaCompra
401	55000	8800	Sandra Roa	28/05/2013
402	45000	7200	Maria Vidal	31/05/2013
403	60000	9600	Maria Vidal	20/08/2013
404	35000	5600	Sandra Roa	20/09/2013

En el siguiente ejemplo se establece a la columna IVA el valor del 1500 y como vendedor "Fabian Ruano" para todas las tuplas de la tabla:

UPDATE Factura
SET IVA = 1500 , Vendedor = 'Fabian Ruano'

Luego de la ejecución la tabla queda así:

4.76. Resultado de actualización de Iva y Vendedor

FACTURA				
Numero	Total	IVA	Vendedor	FechaCompra
401	55000	1500	Fabian Ruano	28/05/2013
402	45000	1500	Fabian Ruano	31/05/2013
403	60000	1500	Fabian Ruano	20/08/2013
404	35000	1500	Fabian Ruano	20/09/2013

En este ejemplo se incrementa el total de la factura en un 10 %:

UPDATE Factura
SET Total = (Total * 1.1)

Luego de la ejecución la tabla queda así:

Tabla 4.77. Resultado de actualización de Total

FACTURA				
Numero	Total	IVA	Vendedor	FechaCompra
401	60500	8800	Sandra Roa	28/05/2013
402	49500	7200	Maria Vidal	31/05/2013
403	66000	9600	Maria Vidal	20/08/2013
404	38500	5600	Sandra Roa	20/09/2013

A continuación se presenta un ejemplo en donde se establece como vendedor a Fabian Ruano a todas las tuplas cuyo IVA supere los 7500 y el Numero de factura este en el listado {401,402}

UPDATE Factura
SET Vendedor = 'Fabian Ruano'
WHERE IVA > 7500 AND Numero IN (401,402)

Luego de la ejecución la tabla queda así:

Tabla 4.78. Resultado actualización con condicional

FACTURA				
Numero	Total	IVA	Vendedor	FechaCompra
401	55000	8800	Fabian Ruano	28/05/2013
402	45000	7200	Maria Vidal	31/05/2013
403	60000	9600	Maria Vidal	20/08/2013
404	35000	5600	Sandra Roa	20/09/2013

Finalmente se puede condicionar la actualización al resultado de sub-consultas, por ejemplo estableciendo la fecha de compra = '2013/11/01' a los registros cuyo número de factura aparezca en la sub-consulta:

```
UPDATE Factura
SET FechaCompra = '2013/11/01'
WHERE Numero IN (
SELECT Numero
FROM Factura
WHERE Vendedor = 'Maria Vidal'
)
```

Luego de la ejecución la tabla queda así:

Tabla 4.79. Resultado de actualización con sub-consultas

FACTURA				
Numero	Total	IVA	Vendedor	FechaCompra
401	55000	8800	Sandra Roa	28/05/2013
402	45000	7200	Maria Vidal	01/11/2013
403	60000	9600	Maria Vidal	01/11/2013
404	35000	5600	Sandra Roa	20/09/2013

4.4 USUARIOS, ROLES Y PRIVILEGIOS

4.4.1 Contextualización

Uno de las principales preocupaciones al momento de diseñar cualquier base de datos es lo relacionado con la seguridad de la misma. Dicha seguridad debe garantizarse en varios niveles:
Red de datos: si los sistemas de bases de datos son distribuidos, es decir, incluyen más de una ubicación física, muy seguramente deberá existir una infraestructura de comunicación entre las diferentes locaciones. Dicha infraestructura puede representar un punto débil en la seguridad del sistema si no se maneja correctamente, debe garantizarse que los mensajes lleguen completos y sin corrupción alguna. Este nivel de seguridad es responsabilidad de los administradores de infraestructura de las organizaciones (generalmente el departamento/área de soporte técnico), y de los proveedores de transferencia de datos si la base de datos así lo requiere.
Físico: este nivel hace referencia a la seguridad física de cada una de las locaciones relacionadas con el sistema de base de datos (única locación en el caso de los sistemas centralizados y más de una en el caso de los distribuidos). La seguridad a este nivel involucra a los diseñadores de las instalaciones, al personal de seguridad y al personal que tiene acceso al hardware de la

organización.
Sistema operativo: además del hardware, es necesario que el software sobre el que se soportan los sistemas gestores de base de datos sea seguro. Los sistemas operativos proveen varios métodos de acceso a su información: interfaz gráfica, escritorio remoto, puertos, servicios, entre otros. Se debe garantizar que dicho software restrinja el acceso de la información solamente a las personas o aplicaciones autorizadas. Usualmente es responsabilidad del departamento de soporte técnico de las organizaciones velar porque los sistemas operativos utilizados y la configuración de los mismos garanticen la seguridad necesaria.
Humano: uno de los niveles que usualmente se descuida es el relacionado con el manejo de información de seguridad por parte de los empleados de las organizaciones. Es necesario que todos los empleados con acceso a los sistemas de bases de datos sean conscientes de las buenas prácticas de seguridad, por ejemplo: creación de contraseñas seguras, modificación periódica de las mismas, gestión de software anti-virus y anti-espías, entre otras. Este nivel suele ser frágil porque depende de todos los usuarios del sistema de base de datos, que pueden llegar a ser muchos o todos los empleados de la organización, deben generarse planes de concientización del problema y de las buenas prácticas.
Sistema Gestor de Base de Datos: además de la línea de defensa que debe proveer el sistema operativo, el SGBD debe garantizar que solamente los usuarios y/o roles autorizados accedan y modifiquen las diferentes tablas, vistas e información que ellas contengan. Cualquier acceso a la base de datos debe ser luego de una autenticación contra el SGBD. El funcionamiento de la seguridad a este nivel es responsabilidad del SGBD, sin embargo, para que este último actúe correctamente el administrador debe encargarse de configurar las restricciones y permisos de acceso. Dado que este documento se enfoca en el diseño y gestión de bases de datos principalmente desde el punto de vista del Sistema Gestor de Bases de Datos, el contenido de esta sección se enfocará en el último nivel de seguridad, es decir, en el relacionado con el SGBD. Los SGBD proveen varios constructos para gestionar y controlar el acceso a las bases de datos con el ánimo de garantizar que las acciones sobre los objetos de la base de datos (metadatos y datos), sólo sean realizados por aquellos usuarios que tienen los permisos para hacerlo. Este sistema de seguridad se basa en permitir o negar acciones sobre los elementos de la base de datos a roles o usuarios.

4.4.2 Usuarios

Un usuario es un elemento de la base de datos que contiene la información necesaria para que un cliente de la base de datos (administrador, programador, usuario final, aplicación,...), se autentique contra el Sistema Gestor de Base de Datos (SGBD), para que éste le asigne los permisos y/o controle las restricciones correspondientes al mismo.

Un usuario se compone básicamente de Nombre y Contraseña. Para crear un usuario en una base de datos se utiliza el siguiente código SQL:
CREATE USER <nombre_usuario> IDENTIFIED BY <contraseña>

Para modificar la contraseña de acceso de un usuario se utiliza el siguiente comando:
ALTER USER <nombre_usuario> IDENTIFIED BY <nueva_contraseña>

Y finalmente para borrar un usuario se puede utilizar el siguiente comando:
DROP USER <nombre_usuario>

Es necesario que quien ejecute los comandos previos posea permisos de administrador sobre la base de datos o en su defecto tenga los privilegios para gestión de usuarios.

4.4.3 Roles

Un Rol (papel), es una abstracción que, entre otras cosas, facilita la gestión de privilegios y restricciones sobre los objetos de una base de datos. Al igual que a los usuarios, a los roles se les definen permisos y restricciones. A un usuario se le pueden asignar uno o más roles, y un rol puede ser asignado a uno o muchos usuarios. Cuando un usuario tiene asignado un rol tiene los mismos privilegios y restricciones del rol.Para definir un nuevo rol se utiliza el siguiente comando:
CREATE ROLE <nombre_rol>

Dado que el rol solamente tiene un nombre, no hay acciones de modificación sobre estos, existe solamente la acción de borrado que tiene la siguiente sintaxis:
DROP ROLE <nombre_rol>.

Para asignar un rol a un usuario se puede utilizar el siguiente constructo SQL:
GRANT <nombre_rol> TO <nombre_usuario>
Para remover un rol a un usuario se puede utilizar el siguiente comando:
REVOKE <nombre_rol> FROM <nombre_usuario>

4.4.4 Privilegios

En esta sección se describirán los privilegios que se pueden conceder a roles o a usuarios y los constructos SQL necesarios para asignarlos. Es de resaltar que estos privilegios pueden establecerse a tablas, vistas o a partes de ambas.
Selección (SELECT): proporciona los permisos de consulta sobre los objetos relacionados.
Inserción (INSERT): permite insertar nuevos registros en los objetos relacionados al privilegio.
Actualización (UPDATE): permite modificar la información contenida en los objetos relacionados.
Borrado (DELETE): permite eliminar registros o tuplas contenidas en los objetos relacionados.
Referencia (REFERENCES): permite crear “constraints” que referencia a la tabla relacionada al privilegio.
Alteración (ALTER): permite realizar modificaciones a la tabla relacionada.
Indexación (INDEX): permite crear índices en la tabla relacionada.
Cabe anotar que la anterior lista de privilegios es la soportada por ORACLE, el listado de posibles privilegios puede variar dependiendo del SGBD.

4.4.5 Asignar y revocar privilegios

En SQL, la sintaxis para la asignación de privilegios a roles o a usuarios para un objeto de la base de datos es la siguiente:
GRANT <privilegio> ON <objeto_bd> TO {<nombre_usuario> | <nombre_rol> }

De otro lado, para remover los privilegios previamente asignados a usuarios o roles se puede utilizar la siguiente sintaxis:
REVOKE <privilegio> ON <objeto_bd> FROM {<nombre_usuario> | <nombre_rol>}

Existe un privilegio especial que le permite conceder dichos privilegios a otros roles o usuarios, para ello es necesario adicionar la sentencia “WITH GRANT OPTION” al final del SQL utilizado para otorgar privilegios. Para remover este privilegio especial es necesario agregar la cláusula “CASCADE” al final del comando para revocar privilegios.
Existe además un constructo que permite establecer todos los privilegios posibles mediante una sola sentencia, este es “ALL PRIVILEGES”.

Incluyendo las tres últimas cláusulas, la sintaxis para asignación de privilegios queda de la siguiente forma:
GRANT {ALL PRIVILEGES | <privilegio> } ON <objeto_bd> TO {<nombre_usuario> | <nombre_rol>} [WITH GRANT OPTION]

Y la sintaxis para remover privilegios así:
REVOKE {ALL PRIVILEGES | <privilegio> } ON <objeto_bd> FROM {<nombre_usuario> | <nombre_rol>} [CASCADE]

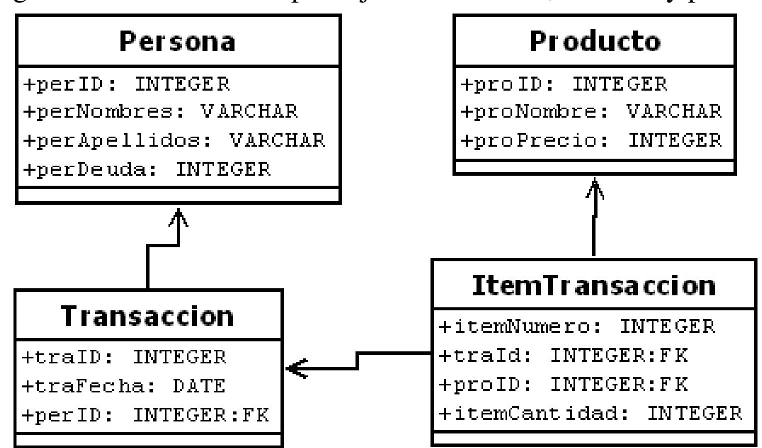
Para que se puedan ejecutar una acción de conceder uno o más privilegios sobre un objeto de la base de datos, quien está ejecutando el procedimiento debe tener, además de los privilegios a otorgar, los permisos para otorgarlos a otros usuarios o roles.

4.4.6 Ejemplo práctico

Planteamiento del Problema

Dada la siguiente base de datos relacional,

Figura 4.5: Base de datos para ejercicio de roles, usuarios y persmisos



Crear el código SQL necesario para realizar las siguientes acciones orientadas a mejorar la seguridad de la base de datos:

i. Crear los siguientes usuarios (nombre / contraseña):
eruano / abcd1234
smroa / mnln4567
mgaona / wxyz9876
ii. Crear los siguientes roles
Administrador
Cliente
iii. Asignar el rol *Administrador* al usuario *eruano*
iv. Asignar el rol *Cliente* a los usuarios *smroa* y *mgaona*
v. Permitir al rol administrador cualquier acción sobre las tablas de la base de datos
vi. Permitir al rol cliente solamente consultar la información de la tabla Persona y de la tabla Producto.
vii. Adicionar al usuario *mgaona* los privilegios para insertar y modificar los registros de la tabla ítem transacción, con los permisos necesarios para conceder dichos privilegios a otros roles o

usuarios

Solución al ejercicio

i.
CREATE USER eruano IDENTIFIED BY abcd1234
CREATE USER smroa IDENTIFIED BY mnln4567
CREATE USER mgaona IDENTIFIED BY wxyz9876
ii.
CREATE ROLE Administrador
CREATE ROLE Cliente
iii.
GRANT Administrador TO eruano
iv.
GRANT Cliente TO smroa
GRANT Cliente TO mgaona
v.
GRANT ALL PRIVILEGES ON Persona TO Administrador
GRANT ALL PRIVILEGES ON Transaccion TO Administrador
GRANT ALL PRIVILEGES ON Item Transaccion TO Administrador
GRANT ALL PRIVILEGES ON Producto TO Administrador
vi.
REVOKE ALL PRIVILEGES ON Persona TO Cliente
REVOKE ALL PRIVILEGES ON Transaccion TO Cliente
REVOKE ALL PRIVILEGES ON Item Transaccion TO Cliente
REVOKE ALL PRIVILEGES ON Producto TO Cliente
GRANT Select ON Persona TO Cliente
GRANT Select ON Producto TO Cliente
vii.
GRANT Insert, Update ON ItemTransaccion TO mgaona WITH GRANT OPTION

5 — APÉNDICE A. Ejemplo de diseño e implementación de una base de datos relacinal en POSGRESQL

5.1 A.1. PLANTEAMIENTO DEL CASO

Se desea crear una base de datos para una organización dedicada a la gestión de recursos forestales, la cual desea gestionar todo el proceso de registro del plan de manejo forestal realizado para el aprovechamiento de especies forestales por personas naturales o jurídicas. Para tal fin se cuenta con la siguiente información:
Con respecto al conjunto de especies forestales o individuos vegetales que controla la organización, cabe resaltar que están organizadas por clases, estas tienen un código, nombre de clasificación, valor en pesos y descripción. Una clase de recurso forestal tiene de una a muchas especies forestales. Las especies forestales tienen un código de especie, nombre común y científico, familia, género y descripción del conjunto de características de la especie. Un especie solo deberá pertenecer a una clasificación.
La organización exige para poder realizar el registro de un plan de manejo forestal primero se deba llevar a cabo una solicitud de aprovechamiento forestal y que esta esté aprobada. Dicha solicitud contiene el código de la solicitud, fecha de registro, área forestal total que se desea aprovechar, edad y tamaño del bosque, finalidad, tiempo de duración del aprovechamiento, numero del predio, estado de la solicitud (aprobada, derogada) y el empleado que la atendió. De los predios se debe gestionar el número catastral, nombre, área y municipio donde está ubicado. Una solicitud deberá incluir solo a un predio y un predio podrá estar incluido en una a muchas solicitudes de aprovechamiento. Con respecto a los interesados en realizar las solicitudes de aprovechamientos, es preciso dejar claro que pueden ser de dos tipos, personas naturales o personas jurídicas. Si es una persona natural se identificará con su número de RUT, nombre, apellidos, teléfono. Si es una persona jurídica se identificará con su número de NIT del negocio, nombre, dirección y municipio donde está ubicado, además de la información de su representante legal, es decir su número de RUT, nombre, apellidos, teléfono. Un interesado realiza de una a muchas solicitudes de aprovechamiento y estas deben pertenecer solo a un interesado.
De los empleados que realizan el trámite de la solicitud se debe gestionar su número de RUT, nombre, apellidos, teléfono, grupo sanguíneo, sexo (Hombre o Mujer). Es importante tener presente que no todos los empleados atienden los tramites, pero una solicitud debe ser tramitada por un empleado.
Las solicitudes de aprovechamiento podrán generar de uno a muchos planes de manejo forestal, de estos se debe controlar el código, número de registro, fecha de ingreso, funcionario que efectúa el trámite del registro, profesional que realizó el plan de manejo forestal, observaciones. Los planes de aprovechamiento contienen de una a muchas especies forestales, de estas se debe estipular la cantidad.
Con respecto al profesional, es preciso especificar que se refiere a un ingeniero forestal, este es el encargado de realizar un plan de manejo forestal a un interesado en realizar un aprovechamiento forestal. Del forestal se debe gestionar número de RUT, tarjeta profesional, nombre, apellidos, teléfono.

5.2 A.2. DISEÑO CONCEPTUAL

Conforme a las necesidades expresadas en el planteamiento del caso, a continuación se elaboró el siguiente E/R como se muestra en la Figura A.1

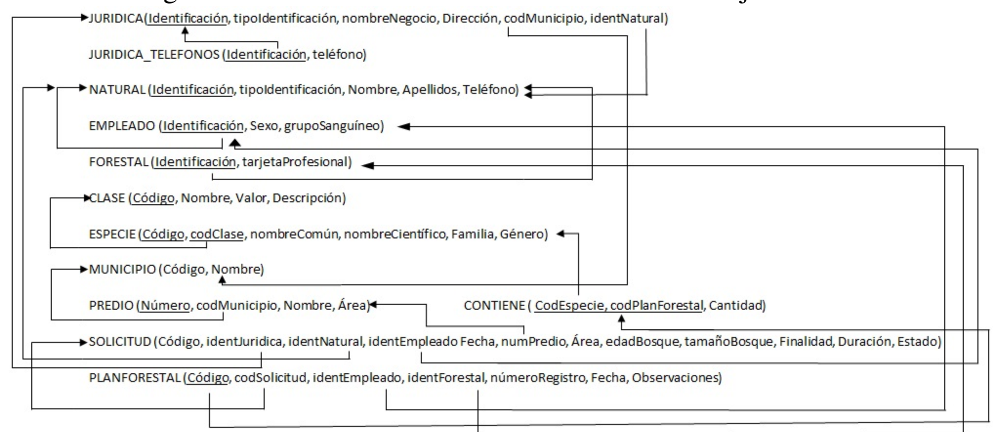
Figura 5.1: A.1. Diagrama E/R para la base de datos del Plan de Manejo Forestal

Identificación
(1,1) PERSONA tipoIdentificación {RUT,NIT}
ES
Código Nombre Valor Descripción CLASE (1,1)
1:N Tiene
(1,n) JURIDICA (0,n) Representa NATURAL (1,1) Nombre Apellidos Teléfono
Teléfonos Dirección nombreNegocio 1:N ES
Código nombreComun nombreCientífico Familia Género (1,n) ESPECIE (1,n)
Realiza 1:N
Sexo grupoSanguineo EMPLEADO (1,1) (1,1) FORESTAL (1,1) tarjetaProfesional
Cantidad Contiene N:M
1:N Tramita
Código númeroRegistro Fecha Observaciones (0,n) (0,n) PLANFORESTAL (0,n) (1,n)
1:N Diseña
1:N Genera
Registrada 1:N
Código Fecha Área edadBosque tamañoBosque Finalidad Duración Estado (Aprobada, Derogada) (1,1) (1,n) SOLICITUD (1,n) (0,n)
Atiende 1:N
(1,1) MUNICIPIO Código Nombre (1,1)
1:N Incluido (1,1)
Número Nombre Área PREDIO (1,n) Ubicado 1:N

5.3 A.3. DISEÑO LÓGICO

Una vez terminada la etapa de análisis y diseño del modelo conceptual, se pasará a la etapa de diseño lógico. Las relaciones resultantes de realizar el proceso de transformación del E/R al Relacional queda resumido en el grafo relacional de la Figura A.2

Figura 5.2: A.2. Grafo Relacional del Plan de Manejo Forestal



5.4 A.4. ESQUEMA FÍSICO DE LA BASE DE DATOS

Creación de Dominios:

CREATE DOMAIN Sexo AS CHAR (6) CHECK (VALUE IN ('Hombre','Mujer'));

CREATE DOMAIN Estado AS CHAR (8) CHECK (VALUE IN ('Aprobada','Derogada'));

CREATE DOMAIN tipoIdentificacion AS CHAR (3) CHECK (VALUE IN ('RUT,'NIT'));

Creación de Tablas:

```
CREATE TABLE CLASE
(
Codigo character(15) NOT NULL,
Nombre character(50) NOT NULL,
Valor money NOT NULL,
Descripcion text NOT NULL,
CONSTRAINT PK_CodigoClase PRIMARY KEY (Codigo)
);

CREATE TABLE ESPECIE
(
Codigo character(15) NOT NULL,
codClase character(15) NOT NULL,
nombreComun character(45) NOT NULL,
nombreCientifico character(45) NOT NULL,
Familia character(45) NOT NULL,
Genero character(45) NOT NULL,
CONSTRAINT PK_ESPECIE PRIMARY KEY (Codigo),
CONSTRAINT FK_CLASE_ESPECIE FOREIGN KEY (codClase)
REFERENCES CLASE (Codigo) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE MUNICIPIO
(
Codigo integer NOT NULL,
Nombre character(30) NOT NULL,
CONSTRAINT PK_MUNICIPIO PRIMARY KEY (Codigo)
);

CREATE TABLE JURIDICA
(
Identificacion integer NOT NULL,
tipoIdentificacion tipoIdentificacion DEFAULT 'NIT' NOT NULL,
nombreNegocio character(100) NOT NULL,
Dirección character(50) NOT NULL,
codMunicipio integer NOT NULL,
identNatural integer NOT NULL,
```

```
CONSTRAINT PK_PERSONA_JURIDICA PRIMARY KEY (Identificacion),
CONSTRAINT FK_MUNICIPIO_JURIDICA FOREIGN KEY (codMunicipio)
REFERENCES MUNICIPIO (Codigo) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT FK_NATURAL_JURIDICA FOREIGN KEY (identNatural)
REFERENCES NATURAL (Identificacion) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE JURIDICA_TELEFONOS
(
Identificacion integer NOT NULL,
Telefono integer NOT NULL,
CONSTRAINT PK_JURIDICA_TELEFONOS PRIMARY KEY (Identificacion, Telefono),
CONSTRAINT JURIDICA_TELEFONOS FOREIGN KEY (Identificacion)
REFERENCES JURIDICA (Identificacion) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE NATURAL
(
Identificacion integer NOT NULL,
tipoIdentificacion tipoIdentificacion DEFAULT ‘RUT’ NOT NULL,
Nombre character(25) NOT NULL,
Apellidos character(25) NOT NULL,
Telefono integer NOT NULL,
CONSTRAINT PK_PERSONA_NATURAL PRIMARY KEY (Identificacion)
);

CREATE TABLE EMPLEADO
(
Identificacion integer NOT NULL,
Sexo Sexo NOT NULL DEFAULT ‘Mujer’ NOT NULL,
grupoSanguineo character(3) NOT NULL,
CONSTRAINT PK_EMPLEADO PRIMARY KEY (Identificacion),
CONSTRAINT FK_PERSONA_NATURAL_EMPLEADO FOREIGN KEY (Identificacion)
REFERENCES NATURAL (Identificacion) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE FORESTAL
(
Identificacion integer NOT NULL,
tarjetaProfesional character(25) NOT NULL,
CONSTRAINT PK_PERSONA_NATURAL_FORESTAL PRIMARY KEY (Identificacion),
CONSTRAINT FK_PERSONA_NATURAL_FORESTAL FOREIGN KEY (Identificacion)
REFERENCES NATURAL (Identificacion) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
);
```

```
CREATE TABLE PREDIO
(
Numero character(25) NOT NULL,
codMunicipio integer NOT NULL,
Nombre character(25) NOT NULL,
Area double precision NOT NULL,
CONSTRAINT PK_PREDIO PRIMARY KEY (Numero),
CONSTRAINT FK_MUNICIPIO_PREDIO FOREIGN KEY (codMunicipio)
REFERENCES MUNICIPIO (Codigo) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE SOLICITUD
(
Codigo integer NOT NULL,
identJuridica integer,
identNatural integer,
identEmpleado integer NOT NULL,
Fecha date NOT NULL,
numPredio character(25) NOT NULL,
Area double precision NOT NULL,
edadBosque integer NOT NULL,
tamanioBosque double precision NOT NULL,
Finalidad character(25) NOT NULL,
Duracion integer NOT NULL,
Estado estado DEFAULT 'Derogada' NOT NULL,
personaNatural boolean,
CONSTRAINT PK_SOLICITUD PRIMARY KEY (Codigo),
CONSTRAINT FK_PERSONA_NATURAL_EMPLEADO_SOLICITUD FOREIGN KEY (iden-
tEmpleado)
REFERENCES EMPLEADO (Identificacion) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT FK_PREDIO_SOLICITUD FOREIGN KEY (numPredio)
REFERENCES PREDIO (Numero) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT PK_PERSONA_JURIDICA_SOLICITUD FOREIGN KEY (identJuridica)
REFERENCES JURIDICA (Identificacion) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT PERSONA_NATURAL_SOLICITUD FOREIGN KEY (identNatural)
REFERENCES NATURAL (Identificacion) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION,
CHECK ((identJuridica IS NOT NULL AND identNatural IS NULL ) OR (identJuridica IS
NULL AND identNatural IS NOT NULL))
);
```

Observación: Especificamos mediante la restricción en el CHECK que la solicitud o la realiza una persona natural o jurídica pero no los dos al mismo tiempo.

```
CREATE TABLE PLANFORESTAL(
Codigo integer NOT NULL,
codSolicitud integer NOT NULL,
identEmpleado integer NOT NULL,
identForestal integer NOT NULL,
numeroRegistro character(25) NOT NULL,
Fecha date NOT NULL,
Observaciones text NOT NULL,
CONSTRAINT PK_PLANFORESTAL PRIMARY KEY (Codigo),
CONSTRAINT FK_PERSONA_NATURAL_EMPLEADO_PLANFORESTAL FOREIGN KEY
(identEmpleado)
REFERENCES EMPLEADO (Identificacion) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT FK_PERSONA_NATURA_FORESTAL FOREIGN KEY (identForestal)
REFERENCES FORESTAL (Identificacion) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT FK_SOLICITUD_PLANFORESTAL FOREIGN KEY (codSolicitud)
REFERENCES SOLICITUD (Codigo) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
);

CREATE TABLE CONTIENE
(
codEspecie character(15) NOT NULL,
codPlanForestal integer NOT NULL,
cantidad double precision NOT NULL,
CONSTRAINT PK_CONTIENE PRIMARY KEY (codEspecie, codPlanForestal),
CONSTRAINT FK_ESPECIE_CONTIENE FOREIGN KEY (codEspecie)
REFERENCES ESPECIE (Codigo) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT PK_PLANFORESTAL_CONTIENE FOREIGN KEY (codPlanForestal)
REFERENCES PLANFORESTAL (Codigo) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
);
```

Creación de Procedimientos almacenados:

```
CREATE OR REPLACE FUNCTION check_codsolicitud()
RETURNS trigger AS '
DECLARE V_Estado SOLICITUD.Estado%type;
BEGIN
– Crear la consulta con el nuevo codSolicitud a ingresar en el Plan de Manejo Forestal
– con el fin de recuperar el estado y realizar la validación
V_Estado=( SELECT SOLICITUD.Estado FROM SOLICITUD WHERE SOLICITUD.Codigo=NEW.codSolicitud);
IF V_Estado="Derogada" THEN
RAISE EXCEPTION "% No se puede realizar el registro del Plan de Manejo Forestal con la
solicitud Derogada", NEW.codSolicitud;
END IF;
RETURN NULL;
```

END;
LANGUAGE 'plpgsql';

Observación: Creamos un procedimiento almacenado que permita validar que al realizar un nuevo registro a la tabla PLANFORESTAL, su solicitud deba estar aprobada. Para esto primero debemos crear una función y luego realizar la asignación definiendo el tipo de procedimiento almacenado a crear.
CREATE TRIGGER check_codsolicitud BEFORE INSERT OR UPDATE ON PLANFORES-TAL
FOR EACH ROW EXECUTE PROCEDURE check_codsolicitud ();

Observación: Asignamos el procedimiento almacenado a la tabla PLANFORESTAL y estipulamos el tipo de TRIGGER que se desea crear, para este caso será BEFORE.

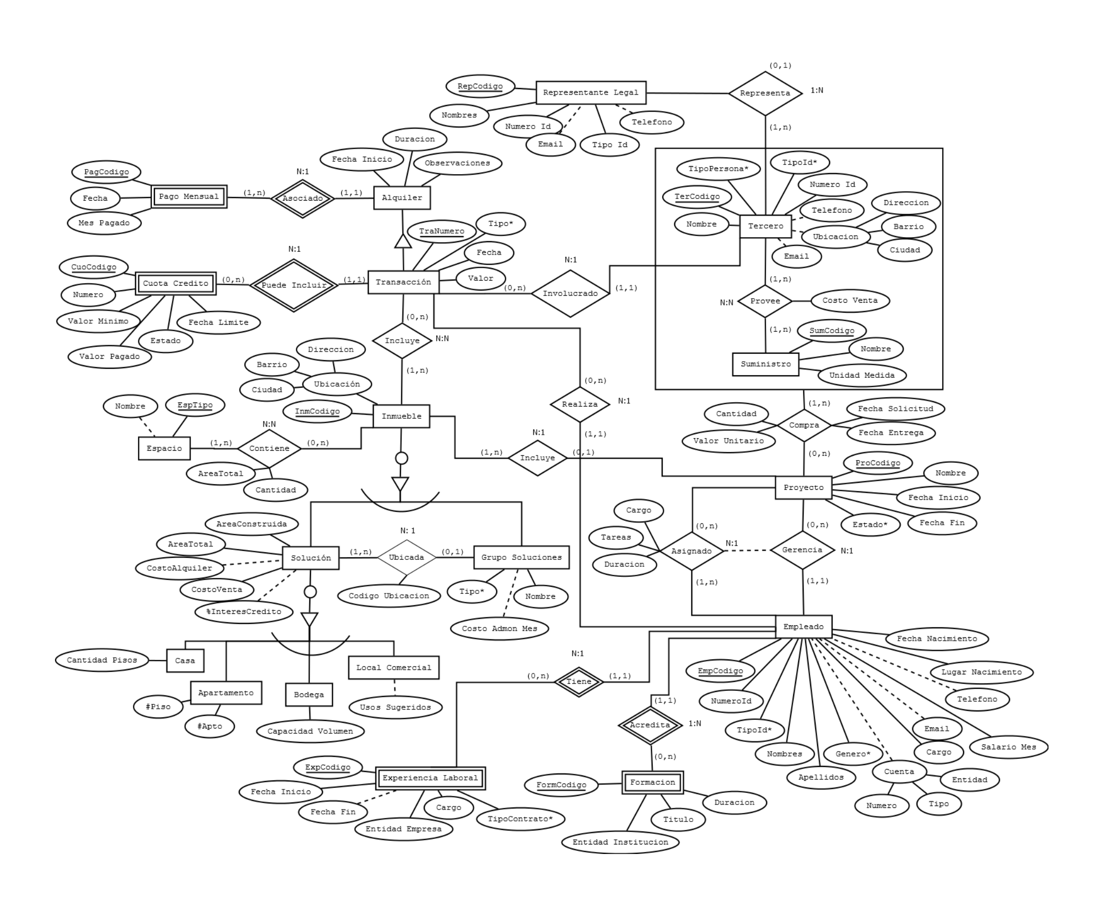
6 — APÉNDICE B. Ejemplo de diseño e implementación de una base de datos relacional en ORACLE

6.1 B.1. PLANTEAMIENTO DEL CASO

La empresa Construimos Ltda., dedicada a la construcción y comercialización de soluciones inmobiliarias requiere que se diseñe e implemente una base de datos que permita almacenar y gestionar toda la información relacionada con sus actividades económicas.
Las soluciones producidas por la empresa incluyen casas, apartamentos, bodegas y locales comerciales, los cuales pueden estar incluidos o no en conjuntos residenciales, condominios u otra estructura. Todos los tipos de solución tienen una ubicación (compuesto por dirección, barrio y ciudad), un número de metros cuadrados construidos, un número de metros cuadrados totales, costo de alquiler y costo de venta.
De los inmuebles tipo casa o apartamento es necesario almacenar la cantidad, el tamaño y el tipo de cada uno de los espacios que contiene, dichos espacios pueden ser de tipo: sala, comedor, alcoba, alcoba principal, alcoba servicio, baño, baño social, baño de lujo, closet, patio, balcón, terraza, cocina, estudio, salón de juegos, sala-comedor, piscina, jardín, antejardín o garaje. Si la casa contiene más de un piso debe almacenarse el número de ellos. En caso de ser un apartamento debe almacenarse el número de piso en el que se encuentra y el número de apartamento asignado. Si la solución se encuentra en un conjunto cerrado, condominio o estructura similar debe almacenarse la información que indique la ubicación dentro de dicho complejo.
De las bodegas se necesita gestionar la cantidad de metros cúbicos de volumen que pueden almacenar, además se requiere conocer el número, tamaño y tipo de espacios que contiene. Sin embargo, a diferencia de lo descrito anteriormente, los tipos pueden ser: espacio almacenamiento, oficina o baño. Para los locales comerciales se requiere almacenar el número, tamaño y tipo de espacios que contiene (espacio genérico o baño), y un campo de texto para sugerencias de uso. En el caso de los conjuntos cerrados, condominios u otra estructura similar que agrupe dos o más soluciones de vivienda y/o locales comerciales, es necesario almacenar la información correspondiente a nombre, tarifa de administración mensual y zonas comunes (cantidad, tamaño y tipo de zona: piscina, zona verde, zona de juegos para niños, gimnasio o salón social).
La empresa incluye gran cantidad y tipo de empleados que van desde vendedores hasta arquitectos e ingenieros. De todos ellos es necesario almacenar tipo de identificación, número de identificación, nombres, apellidos, fecha de nacimiento, lugar de nacimiento, género, teléfonos, correo electrónico, experiencia (cero o muchos registros), estudios (uno o muchos), cargo, asignación salarial mensual, información de cuenta bancaria (tipo de cuenta, número y nombre de banco), información de experiencia laboral (duración, empresa, cargo, tipo contrato) e información de estudios y formación (título, duración, entidad o institución).
Un proyecto de construcción puede incluir desde una única solución inmobiliaria (de cualquier tipo), hasta un conjunto de ellas que pueden estar agrupadas o no en un conjunto cerrado, condominio u otro, de cada uno de ellos se almacena nombre, fecha de inicio, fecha de finalización y estado (planeación, ejecución, cierre, finalizado). Un proyecto tiene un gerente y personal asignado, de esta asignación es necesario almacenar el listado de tareas a cumplir y el tiempo de

asignación. Se requiere además llevar un control de los proveedores y suministros necesarios para la ejecución de los proyectos. Un proveedor puede ser una persona jurídica o una persona natural, de ellos se requiere almacenar nombre, identificación, tipo de identificación, teléfonos, correo electrónico, dirección y si es persona jurídica la información de contacto del representante legal (nombres, identificación, teléfonos, correo electrónico). De los suministros se necesita almacenar nombre, código, unidad de medida y descripción. Cada suministro puede ser provisto por uno o más proveedores (el costo de venta de la unidad de un suministro puede variar dependiendo del proveedor), y a su vez un proveedor puede ofrecer uno o más suministros. Para cada proyecto es necesario almacenar el historial de compras realizadas a cada proveedor, en dicho historial debe detallarse la cantidad de cada suministro adquirido, el precio de compra, la fecha de solicitud y la fecha de entrega.
De los clientes es necesario almacenar número de identificación, tipo de identificación, nombre completo, teléfonos y correo electrónico, en caso que el cliente sea una persona jurídica debe almacenarse información del representante legal (nombres, identificación, teléfonos, correo electrónico).
Un inmueble puede estar disponible para dos tipos de transacciones en la empresa: ventas o alquiler. De cada transacción se requiere almacenar la fecha, tipo de transacción, vendedor, cliente y valor acordado. Un cliente podrá realizar una o más transacciones de diferentes tipos y estas deberán estar relacionadas con diferentes soluciones inmobiliarias. Las ventas pueden ser de contado o crédito, en cualquiera de los casos es necesario almacenar la fecha de transacción y valor total de la misma. Además se deberá tener en cuenta que si esuna venta a crédito se debe aplicar un porcentaje correspondiente al interés por el crédito establecido para cada una de las soluciones inmobiliarias en venta, además se requerirá almacenar un historial de pagos realizados y los pendientes, los cuales deben incluir fecha, valor de pago mínimo y valor de pago realizado, si el pago realizado supera al pago mínimo este valor se descontará en los últimos pagos del crédito y si es necesario se deberán eliminar uno o más pagos finales.
Para las transacciones de tipo Alquiler es necesario gestionar la información correspondiente a fecha de inicio del contrato, duración (en meses) del contrato, valor mensual del alquiler, observaciones al contrato e información de todos los pagos realizados por parte del cliente.

6.2 B.2. DISEÑO CONCEPTUAL

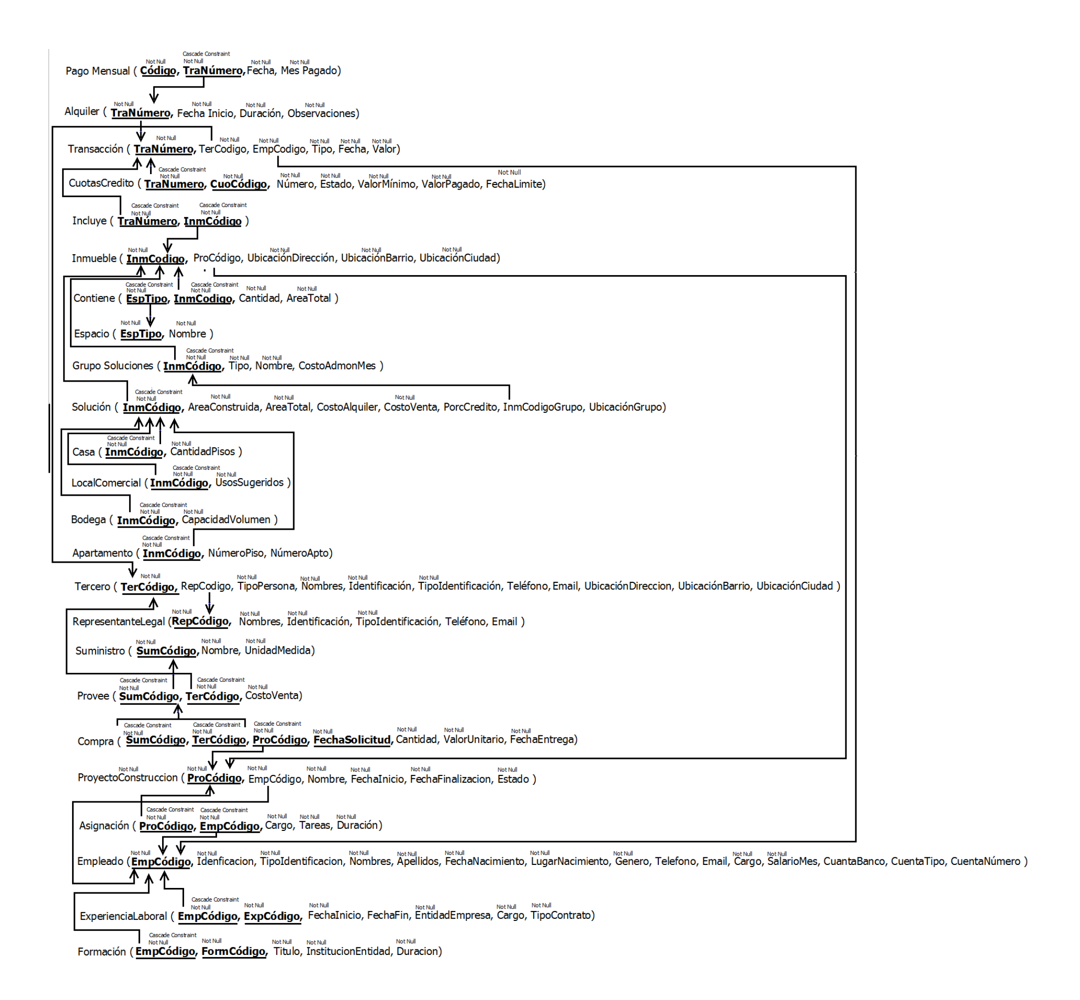


En el modelo se especificaron algunos atributos con asterisco (*) al final del nombre, esto indica que dichos atributos tienen un dominio especifico, a continuación se presenta esa lista de dominios:

- Proyecto.Estado = {Planeación, Ejecución, Cierre, Finalizado}
- Empleado.Genero = {Masculino, Femenino}
- ExperienciaLaboral.TipoContrato = {Fijo Medio Tiempo, Fijo Tiempo Completo, Indefinido Medio Tiempo, Indefinido Tiempo Completo, Prestación de Servicios, Otro}
- GrupoSoluciones.Tipo = {Conjunto Cerrado, Condominio, Centro Comercial, Otro}
- Empleado.TipoId = {Cédula Ciudadanía, Cédula Extranjería, Otro}
- Tercero.TipoId = {Cédula Ciudadanía, Cédula Extranjería, Otro}
- Tercero.TipoPersona = {Natural, Jurídica}
- Transacción.Tipo = {Venta, Alquiler}

En el modelo E-R se incluye una relación Tercero que agrupa tanto los proveedores como los compradores. Se realiza esta simplificación porque para ambos (proveedores y compradores), se requiere almacenar la misma información.

6.3 B.3. DISEÑO LÓGICO



6.4 B.4. IMPLEMENTACION ORACLE

6.4.1 Código SQL correspondiente al modelo relacional de base de datos

```
/*==============================================================*/
/* Table: ALQUILER */
/*==============================================================*/
create table ALQUILER (
TRANUMERO INTEGER not null,
FECHA_INICIO DATE not null,
DURACION VARCHAR2(30) not null,
OBSERVACIONES CLOB not null,
constraint PK_ALQUILER primary key (TRANUMERO)
);

/*==============================================================*/
/* Table: APARTAMENTO */
/*==============================================================*/
create table APARTAMENTO (
INMCODIGO CHAR(20) not null,
NUMERO_PISO INTEGER,
NUMERO_APTO INTEGER,
```

```
constraint PK_APARTAMENTO primary key (INMCODIGO)
);

/*==============================================================*/
/* Table: ASIGNACION */
/*==============================================================*/
create table ASIGNACION (
PROCODIGO CHAR(20) not null,
EMPCODIGO INTEGER not null,
CARGO VARCHAR2(50) not null,
TAREAS CLOB not null,
DURACION VARCHAR2(30) not null,
constraint PK_ASIGNACION primary key (PROCODIGO, EMPCODIGO)
);

/*==============================================================*/
/* Table: BODEGA */
/*==============================================================*/
create table BODEGA (
INMCODIGO CHAR(20) not null,
CAPACIDAD_VOLUMEN FLOAT not null,
constraint PK_BODEGA primary key (INMCODIGO)
);

/*==============================================================*/
/* Table: CASA */
/*==============================================================*/
create table CASA (
INMCODIGO CHAR(20) not null,
CANTIDAD_PISOS INTEGER not null,
constraint PK_CASA primary key (INMCODIGO)
);

/*==============================================================*/
/* Table: COMPRA */
/*==============================================================*/
create table COMPRA (
TERCODIGO CHAR(20) not null,
PROCODIGO CHAR(20) not null,
CODIGO_SUM CHAR(20) not null,
FECHA_SOLICITUD DATE not null,
CANTIDAD INTEGER not null,
VALOR_UNITARIO NUMBER(8,2) not null,
FECHA_ENTREGA DATE not null,
constraint PK_COMPRA primary key (TERCODIGO, PROCODIGO, CODIGO_SUM, FE-
CHA_SOLICITUD)
);
```

```
/*==============================================================*/
/* Table: CONTIENE */
/*==============================================================*/
create table CONTIENE (
ESPTIPO INTEGER not null
constraint CKC_ESPTIPO_CONTIENE check (ESPTIPO in (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
INMCODIGO CHAR(20) not null,
CANTIDAD INTEGER not null,
AREA_TOTAL_ FLOAT not null,
constraint PK_CONTIENE primary key (ESPTIPO, INMCODIGO)
);

/*==============================================================*/
/* Table: CUOTAS_CREDITO */
/*==============================================================*/
create table CUOTAS_CREDITO (
TRANUMERO INTEGER not null,
CUOCODIGO INTEGER not null,
NUMERO INTEGER not null,
ESTADO VARCHAR(20) not null
constraint CKC_ESTADO_CUOTAS_C check (ESTADO in ('Planeacion','Ejecucion','Cierre','Finalizado')),
VALOR_MINIMO NUMBER(8,2) not null,
VALOR_PAGADO NUMBER(8,2) not null,
FECHA_LIMITE DATE not null,
constraint PK_CUOTAS_CREDITO primary key (TRANUMERO, CUOCODIGO)
);

/*==============================================================*/
/* Table: EMPLEADO */
/*==============================================================*/
create table EMPLEADO (
EMPCODIGO INTEGER not null,
IDENTIFICACION_NUMERO INTEGER not null,
IDENTIFICACION_TIPO VARCHAR(20) not null
constraint CKC_IDENTIFICACION_TI_EMPLEADO check (IDENTIFICACION_TIPO in
('Cedula Ciudadania','Cedula Extrangeria')),
NOMBRES VARCHAR2(100) not null,
APELLIDOS VARCHAR2(100) not null,
FECHA_NACIMIENTO DATE not null,
LUGAR_NACIMIENTO VARCHAR2(100) not null,
GENERO VARCHAR(20) not null
constraint CKC_GENERO_EMPLEADO check (GENERO in ('Masculino','Femenino')),
TELEFONOS CLOB,
CORREO_ELECTRONICO VARCHAR2(100),
CARGO VARCHAR2(50) not null,
SALARIO_MES NUMBER(8,2) not null,
CUENTA_BANCO VARCHAR2(50),
CUENTA_TIPO INTEGER,
CUENTA_NUMERO VARCHAR2(20),
```

```
constraint PK_EMPLEADO primary key (EMPCODIGO)
);

/*==============================================================*/
/* Table: ESPACIO */
/*==============================================================*/
create table ESPACIO (
ESPTIPO INTEGER not null
constraint CKC_ESPTIPO_ESPACIO check (ESPTIPO in (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22
NOMBRE VARCHAR2(256),
constraint PK_ESPACIO primary key (ESPTIPO)
);

/*==============================================================*/
/* Table: EXPERIENCIA_LABORAL */
/*==============================================================*/
create table EXPERIENCIA_LABORAL (
EMPCODIGO INTEGER not null,
EXPCODIGO CHAR(10) not null,
FECHA_INICIO DATE not null,
FECHA_FINALIZACION DATE,
ENTIDAD_O_EMPRESA VARCHAR2(100) not null,
CARGO VARCHAR2(50) not null,
TIPO_CONTRATO VARCHAR(20) not null
constraint CKC_TIPO_CONTRATO_EXPERIEN check (TIPO_CONTRATO in (‘Fijo Me-
dio Tiempo’,‘Fijo Tiempo Completo’,‘Indefinido Medio Tiempo’,‘Indefinido Tiempo Comple-
to’,‘Prestacion Servicios’,‘Otro’)),
constraint PK_EXPERIENCIA_LABORAL primary key (EMPCODIGO, EXPCODIGO)
);

/*==============================================================*/
/* Table: FORMACION */
/*==============================================================*/
create table FORMACION (
EMPCODIGO INTEGER not null,
FORMCODIGO VARCHAR2(10) not null,
TITULO VARCHAR2(100) not null,
INSTITUCION_O_ENTIDAD VARCHAR2(100) not null,
DURACION VARCHAR2(30) not null,
constraint PK_FORMACION primary key (EMPCODIGO, FORMCODIGO)
);

/*==============================================================*/
/* Table: GRUPO_SOLUCIONES */
/*==============================================================*/
create table GRUPO_SOLUCIONES (
INMCODIGO CHAR(20) not null,
TIPO VARCHAR(20) not null
constraint CKC_TIPO_GRUPO_SO check (TIPO in (‘Alquiler’,‘Venta’)),
```

```
NOMBRE VARCHAR2(256) not null,
VALOR_ADMON_MES NUMBER(8,2),
constraint PK_GRUPO_SOLUCIONES primary key (INMCODIGO)
);

/*==============================================================*/
/* Table: INCLUYE */
/*==============================================================*/
create table INCLUYE (
TRANUMERO INTEGER not null,
INMCODIGO CHAR(20) not null,
constraint PK_INCLUYE primary key (TRANUMERO, INMCODIGO)
);

/*==============================================================*/
/* Table: INMUEBLE */
/*==============================================================*/
create table INMUEBLE (
INMCODIGO CHAR(20) not null,
PROCODIGO CHAR(20),
UBICACION_DIRECCION VARCHAR(50) not null,
UBICACION_BARRIO VARCHAR(50) not null,
UBICACION_CIUDAD VARCHAR(50) not null,
constraint PK_INMUEBLE primary key (INMCODIGO)
);

/*==============================================================*/
/* Table: LOCAL_COMERCIAL */
/*==============================================================*/
create table LOCAL_COMERCIAL (
INMCODIGO CHAR(20) not null,
USOS_SUGERIDOS CLOB not null,
constraint PK_LOCAL_COMERCIAL primary key (INMCODIGO)
);

/*==============================================================*/
/* Table: PAGO_MENSUAL */
/*==============================================================*/
create table PAGO_MENSUAL (
TRANUMERO INTEGER not null,
PAGCODIGO INTEGER not null,
FECHA DATE not null,
MES_PAGADO VARCHAR2(6) not null,
constraint PK_PAGO_MENSUAL primary key (TRANUMERO, PAGCODIGO)
);
```

```
/*==============================================================*/
/* Table: PROVEE */
/*==============================================================*/
create table PROVEE (
SUMCODIGO CHAR(20) not null,
TERCODIGO CHAR(20) not null,
COSTO_VENTA NUMBER(8,2) not null,
constraint PK_PROVEE primary key (SUMCODIGO, TERCODIGO)
);

/*==============================================================*/
/* Table: PROYECTO_CONSTRUCCION */
/*==============================================================*/
create table PROYECTO_CONSTRUCCION (
PROCODIGO CHAR(20) not null,
EMPCODIGO INTEGER not null,
NOMBRE VARCHAR2(256) not null,
FECHA_INICIO DATE not null,
FECHA_FINALIZACION DATE not null,
ESTADO VARCHAR(20) not null
constraint CKC_ESTADO_PROYECTO check (ESTADO in (‘Planeacion’,‘Ejecucion’,‘Cierre’,‘Finalizado’)),
constraint PK_PROYECTO_CONSTRUCCION primary key (PROCODIGO)
);

/*==============================================================*/
/* Table: REPRESENTANTE_LEGAL */
/*==============================================================*/
create table REPRESENTANTE_LEGAL (
REPCODIGO CHAR(20) not null,
NOMBRES VARCHAR2(100) not null,
IDENTIFICACION INTEGER not null,
TIPO_IDENTIFICACION VARCHAR(20) not null
constraint CKC_TIPO_IDENTIFICACI_REPRESEN check (TIPO_IDENTIFICACION in (‘Ce-
dula Ciudadania’,‘Cedula Extrangeria’)),
TELEFONOS_ CLOB,
CORREO_ELECTRONICO VARCHAR2(100),
constraint PK_REPRESENTANTE_LEGAL primary key (REPCODIGO)
);

/*==============================================================*/
/* Table: SOLUCION */
/*==============================================================*/
create table SOLUCION (
INMCODIGO CHAR(20) not null,
AREA_CONSTRUIDA FLOAT not null,
AREA_TOTAL_ FLOAT not null,
COSTO_ALQUILER NUMBER(8,2),
COSTO_VENTA NUMBER(8,2) not null,
PORCENTAJE_CREDITO FLOAT,
```

```
INMCODIGOGRUPO CHAR(20),
CODIGO_UBICACION VARCHAR(20),
constraint PK_SOLUCION primary key (INMCODIGO)
);

/*==============================================================*/
/* Table: SUMINISTRO */
/*==============================================================*/
create table SUMINISTRO (
SUMCODIGO CHAR(20) not null,
NOMBRE VARCHAR2(256) not null,
UNIDAD_MEDIDA CHAR(10) not null,
constraint PK_SUMINISTRO primary key (SUMCODIGO)
);

/*==============================================================*/
/* Table: TERCERO */
/*==============================================================*/
create table TERCERO (
TERCODIGO CHAR(20) not null,
REPCODIGO CHAR(20),
TIPO_PERSONA VARCHAR(20) not null
constraint CKC_TIPO_PERSONA_TERCERO check (TIPO_PERSONA in ('Natural','Juridica')),
NOMBRE VARCHAR2(256) not null,
TIPO_IDENTIFICACION_ VARCHAR(20) not null
constraint CKC_TIPO_IDENTIFICACI_TERCERO check (TIPO_IDENTIFICACION_ in ('Ce-
dula Ciudadania','Cedula Extrangeria')),
NUMERO_IDENTIFICACION_ INTEGER not null,
TELEFONOS CLOB,
CORREO_ELECTRONICO VARCHAR2(100),
UBICACION_DIRECCION VARCHAR2(50),
UBICACION_BARRIO VARCHAR2(50),
UBICACION_CIUDAD VARCHAR2(50),
constraint PK_TERCERO primary key (TERCODIGO)
);

/*==============================================================*/
/* Table: TRANSACCION */
/*==============================================================*/
create table TRANSACCION (
TRANUMERO INTEGER not null,
TERCODIGO CHAR(20) not null,
EMPCODIGO INTEGER not null,
TIPO VARCHAR(20) not null
constraint CKC_TIPO_TRANSACC check (TIPO in ('Alquiler','Venta')),
FECHA DATE not null,
VALOR NUMBER(8,2) not null,
constraint PK_TRANSACCION primary key (TRANUMERO)
);
```

```
alter table ALQUILER
add constraint FK_ALQUILER_ES___TRANSACC foreign key (TRANUMERO)
references TRANSACCION (TRANUMERO);

alter table APARTAMENTO
add constraint FK_APARTAME_INHERITAN_SOLUCION foreign key (INMCODIGO)
references SOLUCION (INMCODIGO);

alter table ASIGNACION
add constraint FK_ASIGNACI_ASIGNACIO_PROYECTO foreign key (PROCODIGO)
references PROYECTO_CONSTRUCCION (PROCODIGO);

alter table ASIGNACION
add constraint FK_ASIGNACI_ASIGNACIO_EMPLEADO foreign key (EMPCODIGO)
references EMPLEADO (EMPCODIGO);

alter table BODEGA
add constraint FK_BODEGA_INHERITAN_SOLUCION foreign key (INMCODIGO)
references SOLUCION (INMCODIGO);

alter table CASA
add constraint FK_CASA_INHERITAN_SOLUCION foreign key (INMCODIGO)
references SOLUCION (INMCODIGO);

alter table COMPRA
add constraint FK_COMPRA_COMPRA_PROVEE foreign key (CODIGO_SUM, TERCODI-
GO)
references PROVEE (SUMCODIGO, TERCODIGO);

alter table COMPRA
add constraint FK_COMPRA_COMPRA2_PROYECTO foreign key (PROCODIGO)
references PROYECTO_CONSTRUCCION (PROCODIGO);

alter table CONTIENE
add constraint FK_CONTIENE_CONTIENE_ESPACIO foreign key (ESPTIPO)
references ESPACIO (ESPTIPO);

alter table CONTIENE
add constraint FK_CONTIENE_CONTIENE2_INMUEBLE foreign key (INMCODIGO)
references INMUEBLE (INMCODIGO);

alter table CUOTAS_CREDITO
add constraint FK_CUOTAS_C_PUEDE_INC_TRANSACC foreign key (TRANUMERO)
references TRANSACCION (TRANUMERO);

alter table EXPERIENCIA_LABORAL
add constraint FK_EXPERIEN_TIENE_EMPLEADO foreign key (EMPCODIGO)
references EMPLEADO (EMPCODIGO);
```

alter table FORMACION
add constraint FK_FORMACIO_ACREDITA_EMPLEADO foreign key (EMPCODIGO)
references EMPLEADO (EMPCODIGO);

alter table GRUPO_SOLUCIONES
add constraint FK_GRUPO_SO_ES2_INMUEBLE foreign key (INMCODIGO)
references INMUEBLE (INMCODIGO);

alter table INCLUYE
add constraint FK_INCLUYE_INCLUYE_TRANSACC foreign key (TRANUMERO)
references TRANSACCION (TRANUMERO);

alter table INCLUYE
add constraint FK_INCLUYE_INCLUYE2_INMUEBLE foreign key (INMCODIGO)
references INMUEBLE (INMCODIGO);

alter table INMUEBLE
add constraint FK_INMUEBLE_INCLUYE__PROYECTO foreign key (PROCODIGO)
references PROYECTO_CONSTRUCCION (PROCODIGO);

alter table LOCAL_COMERCIAL
add constraint FK_LOCAL_CO_INHERITAN_SOLUCION foreign key (INMCODIGO)
references SOLUCION (INMCODIGO);

alter table PAGO_MENSUAL
add constraint FK_PAGO_MEN_RELACIONA_ALQUILER foreign key (TRANUMERO)
references ALQUILER (TRANUMERO);

alter table PROVEE
add constraint FK_PROVEE_PROVEE_SUMINIST foreign key (SUMCODIGO)
references SUMINISTRO (SUMCODIGO);

alter table PROVEE
add constraint FK_PROVEE_PROVEE2_TERCERO foreign key (TERCODIGO)
references TERCERO (TERCODIGO);

alter table PROYECTO_CONSTRUCCION
add constraint FK_PROYECTO_GERENCIA_EMPLEADO foreign key (EMPCODIGO)
references EMPLEADO (EMPCODIGO);

alter table SOLUCION
add constraint FK_SOLUCION_ES_INMUEBLE foreign key (INMCODIGO)
references INMUEBLE (INMCODIGO);

alter table SOLUCION
add constraint FK_SOLUCION_REFERENCE_GRUPO_SO foreign key (INMCODIGOGRU-
PO)
references GRUPO_SOLUCIONES (INMCODIGO);

```
alter table TERCERO
add constraint FK_TERCERO_REPRESENT_REPRESEN foreign key (REPCODIGO)
references REPRESENTANTE_LEGAL (REPCODIGO);

alter table TRANSACCION
add constraint FK_TRANSACC_INVOLUCRA_TERCERO foreign key (TERCODIGO)
references TERCERO (TERCODIGO);

alter table TRANSACCION
add constraint FK_TRANSACC_REALIZA_EMPLEADO foreign key (EMPCODIGO)
references EMPLEADO (EMPCODIGO);
```

6.4.2 Código SQL para manejar restricciones propias del problema modelado

Restricción de herencia disjunta entre las relaciones casa, apartamento, bodega y local comercial

Es necesario garantizar que una misma solución no está registrada en más de un tipo, es decir, garantizar que si se encuentra registrada como Casa no esté en ninguno de los otros tipos (Apto, Bodega o Local), y así con todos los tipos de solución considerados.Para ello presentamos dos posibles vías para garantizar lo requerido:

Solución 1: Establecer un check en cada una de las relaciones "hijo" que valide que el Código Inmueble no exista en ninguna otra de las relaciones "hijo".

```
ALTER TABLE Casa ADD CONSTRAINT chk_SolUnicaCasa CHECK
(InmCodigo NOT IN
(Select Distinct CODIGO_INMUEBLE FROM Apartamento, Bodega, Local_Comercial )
);
ALTER TABLE Apartamento ADD CONSTRAINT chk_SolUnicaApto CHECK
(InmCodigo NOT IN
(Select Distinct CODIGO_INMUEBLE FROM Casa, Bodega, Local_Comercial )
);
ALTER TABLE Bodega ADD CONSTRAINT chk_SolUnicaBodega CHECK
(InmCodigo NOT IN
(Select Distinct CODIGO_INMUEBLE FROM Casa, Apartamento, Local_Comercial )
);
ALTER TABLE Local_Comercial ADD CONSTRAINT chk_SolUnicaLocal CHECK
(InmCodigo NOT IN
(Select Distinct CODIGO_INMUEBLE FROM Casa, Apartamento, Bodega)
);
```

Solución 2: Crear un trigger para cada una de las relaciones "hijo" que se dispare al momento de insertar tuplas y que lance una excepción en caso de que el "Código Inmueble" exista previamente en las otras relaciones "hijo".

```
CREATE OR REPLACE TRIGGER tg_AddCasa BEFORE INSERT ON Casa FOR EACH
ROW
DECLARE
MiException EXCEPTION;
```

```
iContador INTEGER;
BEGIN
SELECT Count(*) INTO iContador FROM (
SELECT InmCodigo FROM Apartamento UNION
SELECT InmCodigo FROM Bodega UNION SELECT InmCodigo FROM Local_Comercial )
Alfa
WHERE Alfa.InmCodigo = :New.InmCodigo ;
If iContador > 0 THEN RAISE MiException; END IF
END;
CREATE OR REPLACE TRIGGER tg_AddApartamento BEFORE INSERT ON Apartamento
FOR EACH ROW
DECLARE
MiException EXCEPTION;
iContador INTEGER;
BEGIN
SELECT Count(*) INTO iContador FROM ( SELECT InmCodigo FROM Casa UNION
SELECT InmCodigo FROM Bodega UNION SELECT InmCodigo FROM Local_Comercial )
Alfa
WHERE Alfa.InmCodigo = :New.InmCodigo ;
If iContador > 0 THEN RAISE MiException; END IF
END;
CREATE OR REPLACE TRIGGER tg_AddBodega BEFORE INSERT ON Bodega FOR EACH
ROW
DECLARE
MiException EXCEPTION;
iContador INTEGER;
BEGIN
SELECT Count(*) INTO iContador FROM ( SELECT InmCodigo FROM Casa UNION
SELECT InmCodigo FROM Apartamento UNION SELECT InmCodigo FROM Local_Comercial
) Alfa
WHERE Alfa.InmCodigo = :New.InmCodigo ;
If iContador > 0 THEN RAISE MiException; END IF
END;
CREATE OR REPLACE TRIGGER tg_AddLocal BEFORE INSERT ON Local_Comercial
FOR EACH ROW
DECLARE
MiException EXCEPTION;
iContador INTEGER;
BEGIN
SELECT Count(*) INTO iContador FROM ( SELECT InmCodigo FROM Casa UNION
SELECT InmCodigo FROM Apartamento UNION SELECT InmCodigo FROM Bodega
) Alfa
WHERE Alfa.InmCodigo = :New.InmCodigo ;
If iContador > 0 THEN RAISE MiException; END IF
END;
```

Restricciones de herencia disjunta entre las relaciones Inmueble, Solución y Grupo_Soluciones

Es necesario garantizar que un mismo inmueble no este registrado en más de un tipo, es decir, garantizar que si se encuentra registrado como Solución no esté registrado como "Grupo Soluciones" y viceversa. Al igual que en el caso anterior se presentan dos posibles soluciones a

la problemática:

Solución 1: Establecer un check en cada una de las relaciones "hijo" que valide que el Código Inmueble no exista en ninguna otra de las relaciones "hijo".

```
ALTER TABLE Solucion ADD CONSTRAINT chk_SolUnicaSolucion CHECK
(InmCodigo NOT IN
(Select Distinct InmCodigo FROM Grupo_Soluciones )
);
ALTER TABLE Grupo_Soluciones ADD CONSTRAINT chk_SolUnicaGrupo CHECK
(InmCodigo NOT IN
(Select Distinct InmCodigo FROM Solucion )
);
```

Solución 2: Crear un trigger para cada una de las relaciones "hijo" que se dispare al momento de insertar tuplas y que lance una excepción en caso de que el "Código Inmueble" exista previamente en las otras relaciones "hijo".

```
CREATE OR REPLACE TRIGGER tg_AddSol BEFORE INSERT ON Solucion FOR EACH
ROW
DECLARE
MiException EXCEPTION;
iContador INTEGER;
BEGIN
SELECT Count(*) INTO iContador FROM Grupo_Soluciones
WHERE Grupo_Soluciones.InmCodigo = :New.InmCodigo ;
If iContador > 0 THEN RAISE MiException; END IF
END;
CREATE OR REPLACE TRIGGER tg_AddGrpSol BEFORE INSERT ON Grupo_Soluciones
FOR EACH ROW
DECLARE
MiException EXCEPTION;
iContador INTEGER;
BEGIN
SELECT Count(*) INTO iContador FROM Solucion
WHERE Solucion.InmCodigo = :New.InmCodigo ;
If iContador > 0 THEN RAISE MiException; END IF
If iContador > 0 THEN RAISE MiException; END IF
END;
```

Restricciones de Área Total menor o igual a Área Construida

Es necesario garantizar que el valor registrado en el área construida sea siempre menor o igual al área total de la solución. La solución incluye modificar la tabla solución para agregar un check que garantice que el área total sea mayor o igual que el área construida.ALTER TABLER Solucion ADD CONSTRAINT ChkAreasSolucion CHECK (Area_Construida <= Area_Total);

Restricciones Área Total igual a suma de Áreas Individuales

Es necesario garantizar que tras una actualización del inventario de espacios de un inmueble de tipo "Solución", se actualice el campo "Área Total" del mismo con el resultado de sumar las Áreas Totales de los espacios que este contiene. La solución aquí planteada es crear un trigger que actualice el valor del Area Total de una solución cuando se modifique el registro de espacios

que esta contiene.

```
CREATE OR REPLACE TRIGGER tg_UpdateAreaTotal
AFTER INSERT OR UPDATE ON Contiene
FOR EACH ROW
DECLARE
iTotalArea INTEGER;
iContador INTEGER;
BEGIN
SELECT Count(*) INTO iContador FROM Solucion
WHERE InmCodigo = :New.InmCodigo ;
IF iContador > 0 THEN
SELECT SUM(Area_Total) INTO iTotalArea FROM Contiene
WHERE InmCodigo = :New.InmCodigo ;
UPDATE Solucion SET Area_Total = iTotalArea
WHERE InmCodigo = :New.InmCodigo ;
END IF
END;
```

Restricción de Exclusión

Es necesario controlar que si se asigna un empleado a un proyecto éste no figure como el gerente del mismo, y viceversa. La solución aquí propuesta incluye la creación de dos triggers, uno para controlar la inserción o actualización de los gerentes de proyecto y otro para controlar la inserción o actualización de los empleados asignados a un proyecto:

```
CREATE OR REPLACE TRIGGER tg_AddGerente BEFORE INSERT OR UPDATE ON
Proyecto FOR EACH ROW
DECLARE
MiException EXCEPTION;
iContador INTEGER;
BEGIN
SELECT Count(*) INTO iContador FROM Asignacion
WHERE ProCodigo = :NEW.ProCodigo AND EmpCodigo = :NEW.EmpCodigo ;
If iContador > 0 THEN RAISE MiException; END IF
END;
CREATE OR REPLACE TRIGGER tg_AsignaEmpleado BEFORE INSERT OR UPDATE ON
Asignacion
FOR EACH ROW
DECLARE
MiException EXCEPTION;
iGerente Proyecto.EmpCodigo%TYPE;
BEGIN
SELECT EmpCodigo INTO iGerente FROM Proyecto
WHERE ProCodigo = :NEW.ProCodigo ;
If :NEW.EmpCodigo = iGerente THEN RAISE MiException; END IF
END;
```

7 — APENDICE C. Ejemplo de diseño e implementación de una base de datos relacional

7.1 C.1. PLANTEAMIENTO DEL CASO

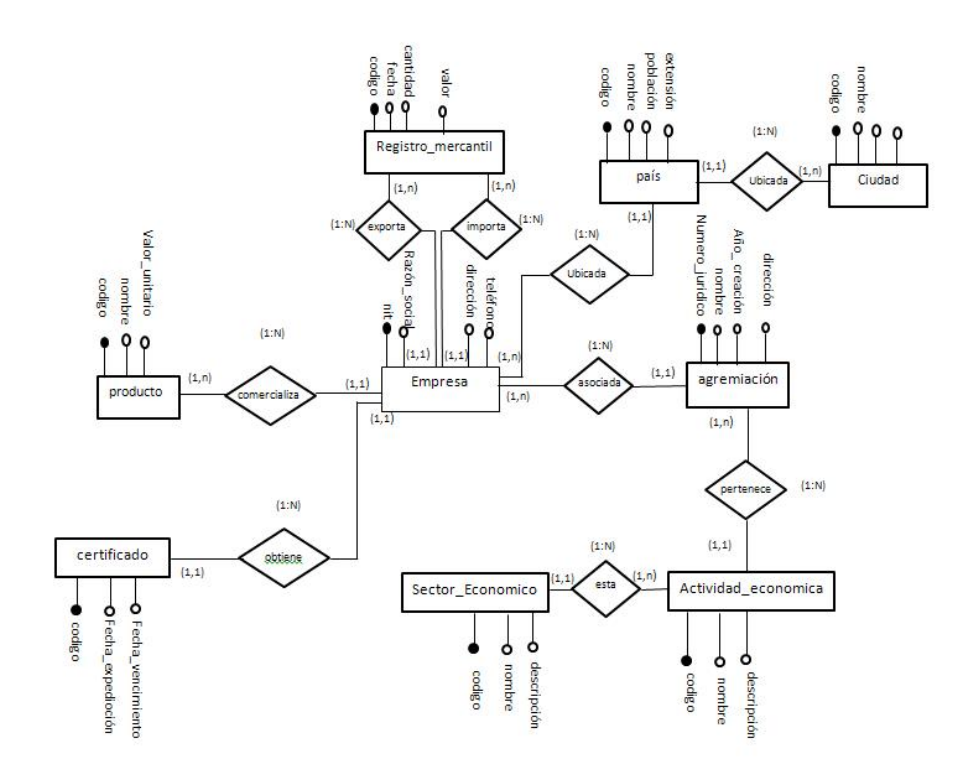
Un sector es una parte de la actividad económica cuyos elementos tienen características comunes, guardan una unidad y se diferencian de otras agrupaciones. Por ejemplo en Colombia existe el sector primario o agropecuario que es el sector que obtiene el producto de sus actividades directamente de la naturaleza, sin ningún proceso de transformación. Dentro de este sector se encuentran la agricultura, la ganadería, la silvicultura, la caza y la pesca. En Colombia existen otros sectores como el secundario o industrial, el terciario o de servicios, entre otros. Cada sector desarrolla actividades económicas que se diferencien aún más dependiendo de su especialización. Una empresa es una unidad productiva dedicada y organizada para la explotación de una actividad económica razón por la que pertenece a un sector económico. Una empresa solo pertenece a un sector económico, pero se puede dedicar a varias actividades económicas que se encuentren clasificadas en ese sector. Un sector económico cuenta con varias organizaciones o agremiaciones a las que se puede asociar una empresa. Por regla general una empresa solo se puede asociar a un gremio dentro del sector.

Toda empresa ofrece un catalogo de productos o servicios, los cuales se pueden exportar a diferentes países, en el marco de tratados como el TLC: Las exportaciones de una empresa se mide por el valor total en dólares de las exportaciones mensuales realizadas a un país. Esta medición se hace por producto. Se debe registrar la información de la empresa que exporta y la que importa.

Una empresa para poder exportar ciertos productos debe tener autorizaciones previas o certificados expedidos por diversas instituciones gubernamentales o de control. Son ejemplos de estos certificados: Certificados Sanitarios, Certificados de Calidad, entre otros.

Con el objeto de mantener actualizada la información económica del país se le solicita que diseñe una base de datos relacional que almacene la información concerniente a las actividades económicas de las empresas y sus exportaciones. Se requiere almacenar la siguiente información. De las empresas se requiere almacenar el NIT, razón social, dirección, teléfono, ciudad y país donde se encuentra la sede principal. También se debe almacenar la información de los certificados o autorizaciones de exportación. De estos últimos se debe registrar el tipo de certificación, fecha de expedición, fecha de vencimiento, restricciones y la entidad gubernamental que la emite. Del sector económico se debe almacenar el nombre o denominación y una descripción general del sector y todas las actividades económicas que pertenecen o forman el sector. Una actividad económica esta caracterizada por el nombre de la actividad y la descripción de la misma. De una agremiación se debe almacenar su número de su Personería Jurídica, nombre, año de fundación o creación, dirección, ciudad y país donde se encuentra ubicada la sede principal. Con respecto la información de los Países se debe registrar su nombre, población, extensión y región a la cual pertenece. De una exportación se debe registrar la empresa de origen (Exportadora) y la empresa de destino (importadora), el producto, la cantidad de producto, el valor y fecha de la exportación.

7.2 C.2. DISEÑO CONCEPTUAL



7.3 C.3. DISEÑO LOGICO



Ciudad (código, nombre)

País (código, nombre, población, extensión, código_c)

Registro_mercantil (código, fecha, cantidad, valor, nit1, nit2)

Producto (código_producto, nombre, valor_unitario, nit)

Empresa (nit, razón_social, dirección, teléfono, código_certificado, numero_juridico, código_p)

Certificado (código, fecha_expedicion, fecha_vencimiento)

Agremiación (numero_juridico, nombre, ano_creacion, dirección, código_actividad)

Actividad_economica (código_actividad , nombre, descripción, código_sector)

Sector_Economico (código_sector, nombre, descripción)

7.4 C.4. IMPLEMENTACION

```
******************************************************************
CREATE
TABLE ACTIVIDAD_ECONOMICA
(
ACT_Codigo INTEGER NOT NULL ,
ACT_Nombre VARCHAR2 (50) ,
ACT_Descripcion VARCHAR2 (50) ,
SEC_Codigo INTEGER
) ;
ALTER TABLE ACTIVIDAD_ECONOMICA ADD CONSTRAINT ACTIVIDAD_ECONOMICA_PK
PRIMARY
KEY
(
ACT_Codigo
)
;
******************************************************************
CREATE
TABLE AGREMIACION
(
AGR_Numero_Juridico INTEGER NOT NULL ,
AGR_Nombre VARCHAR2 (50) ,
AGR_Año_Creacion DATE ,
AGR_Direccion VARCHAR2 (50) ,
ACT_Codigo INTEGER
) ;
ALTER TABLE AGREMIACION ADD CONSTRAINT AGREMIACION_PK PRIMARY KEY
(
AGR_Numero_Juridico
)
;
******************************************************************
CREATE
TABLE CERTIFICADO
(
CER_Codigo INTEGER NOT NULL ,
CER_Fecha_Expedicion DATE ,
CER_Fecha_Vencimiento DATE
) ;
ALTER TABLE CERTIFICADO ADD CONSTRAINT CERTIFICADO_PK PRIMARY KEY
(
CER_Codigo
)
;
```

```
*******************************************************************
CREATE
TABLE CIUDAD
(
CIU_Codigo INTEGER NOT NULL ,
CIU_Nombre VARCHAR2 (50) ,
PAI_Codigo INTEGER NOT NULL
) ;
*******************************************************************
ALTER TABLE CIUDAD ADD CONSTRAINT CIUDAD_PK PRIMARY KEY
(
CIU_Codigo, PAI_Codigo
)
;
*******************************************************************
CREATE
TABLE EMPRESA
(
EMP_NIT VARCHAR2 (20) NOT NULL ,
EMP_Razon_Social VARCHAR2 (50) ,
EMP_Direccion VARCHAR2 (50) ,
EMP_Telefono CHAR (10) ,
CER_Codigo INTEGER ,
AGR_Numero_Juridico INTEGER
) ;
*******************************************************************
ALTER TABLE EMPRESA ADD CONSTRAINT EMPRESA_PK PRIMARY KEY
(
EMP_NIT
)
;
*******************************************************************
CREATE
TABLE PAIS
(
PAI_Codigo INTEGER NOT NULL ,
PAI_Nombre VARCHAR2 (50) ,
PAI_Poblacion VARCHAR2 (50) ,
PAI_Extension VARCHAR2 (50) ,
PAI_Region VARCHAR2 (50)
) ;
*******************************************************************
ALTER TABLE PAIS ADD CONSTRAINT PAIS_PK PRIMARY KEY
(
PAI_Codigo
)
;
```

```
*******************************************************************
CREATE
TABLE PRODUCTO
(
PRO_Codigo INTEGER NOT NULL ,
PRO_Nombre VARCHAR2 (50) ,
PRO_Valor_Unitario NUMBER ,
EMP_NIT VARCHAR2 (20)
) ;
*******************************************************************
ALTER TABLE PRODUCTO ADD CONSTRAINT PRODUCTO_PK PRIMARY KEY
(
PRO_Codigo
)
;
*******************************************************************
CREATE
TABLE Registro_mercantil
(
COM_Codigo UNKNOWN
– ERROR: Datatype UNKNOWN is not allowed
NOT NULL ,
COM_Fecha DATE ,
COM_Cantidad INTEGER ,
COM_Valor NUMBER ,
EMP_NIT VARCHAR2 (20) ,
EMP_NIT1 VARCHAR2 (20) ,
PRO_Codigo INTEGER
) ;
*******************************************************************
ALTER TABLE Registro_mercantil ADD CONSTRAINT Registro_mercantil_PK PRIMARY
KEY
(
COM_Codigo
)
;
*******************************************************************
CREATE
TABLE SECTOR_ECONOMICO
(
SEC_Codigo INTEGER NOT NULL ,
SEC_Nombre VARCHAR2 (50) ,
SEC_Descripcion VARCHAR2 (50) ,
PAI_Codigo INTEGER
) ;
*******************************************************************
ALTER TABLE SECTOR_ECONOMICO ADD CONSTRAINT SECTOR_ECONOMICO_PK
PRIMARY KEY
(
```

SEC_Codigo
)
;
****

ALTER TABLE ACTIVIDAD_ECONOMICA ADD CONSTRAINT
ACTIVIDAD_ECONOMICA_SECTOR_ECONOMICO_FK FOREIGN KEY
(
SEC_Codigo
)
REFERENCES SECTOR_ECONOMICO
(
SEC_Codigo
)
;
****

ALTER TABLE AGREMIACION ADD CONSTRAINT AGREMIACION_ACTIVIDAD_ECONOMICA_FK
FOREIGN KEY
(
ACT_Codigo
)
REFERENCES ACTIVIDAD_ECONOMICA
(
ACT_Codigo
)
;
****
ALTER TABLE CIUDAD ADD CONSTRAINT CIUDAD_PAIS_FK FOREIGN KEY
(
PAI_Codigo
)
REFERENCES PAIS
(
PAI_Codigo
)
;
****
ALTER TABLE EMPRESA ADD CONSTRAINT EMPRESA_AGREMIACION_FK FOREIGN KEY
(
AGR_Numero_Juridico
)
REFERENCES AGREMIACION
(
AGR_Numero_Juridico
)
;

****

ALTER TABLE EMPRESA ADD CONSTRAINT EMPRESA_CERTIFICADO_FK FOREIGN KEY
(
CER_Codigo
)
REFERENCES CERTIFICADO
(
CER_Codigo
)
;

****

ALTER TABLE PRODUCTO ADD CONSTRAINT PRODUCTO_EMPRESA_FK FOREIGN KEY
(
EMP_NIT
)
REFERENCES EMPRESA
(
EMP_NIT
)
;

****

ALTER TABLE Registro_mercantil ADD CONSTRAINT Registro_mercantil_EMPRESA_FK FOREIGN KEY
(
EMP_NIT
)
REFERENCES EMPRESA
(
EMP_NIT
)
;

****

ALTER TABLE Registro_mercantil ADD CONSTRAINT Registro_mercantil_EMPRESA_FKv1 FOREIGN KEY
(
EMP_NIT1
)
REFERENCES EMPRESA
(
EMP_NIT
)
;

****

ALTER TABLE Registro_mercantil ADD CONSTRAINT Registro_mercantil_PRODUCTO_FK FOREIGN KEY
(

```
PRO_Codigo
)
REFERENCES PRODUCTO
(
PRO_Codigo
)
;
*******************************************************************
ALTER TABLE SECTOR_ECONOMICO ADD CONSTRAINT SECTOR_ECONOMICO_PAIS_FK
FOREIGN
KEY
(
PAI_Codigo
)
REFERENCES PAIS
(
PAI_Codigo
)
;
*******************************************************************
```



Edición: Marzo de 2014.
Este texto forma parte de la Iniciativa Latinoamericana de Libros de Texto abiertos (LATIn), proyecto financiado por la Unión Europea en el marco de su Programa ALFA III EuropeAid.

